HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

TOÁN RỜI RẠC 2

NGUYỄN DUY PHƯƠNG

Hà Nội 2016

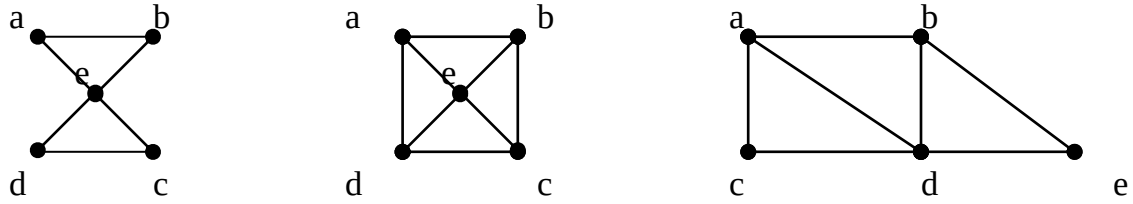
CHƯƠNG 4. ĐỒ THỊ EULER, ĐỒ THỊ HAMIL TON

4.1. Đồ thị Euler, đồ thị nửa Euler

Định nghĩa. Chu trình đơn trong đồ thị G đi qua mỗi cạnh của đồ thị đúng một lần được gọi là chu trình Euler. Đường đi đơn trong G đi qua mỗi cạnh của nó đúng một lần được gọi là đường đi Euler. Đồ thị được gọi là đồ thị Euler nếu nó có chu trình Euler. Đồ thị có đường đi Euler được gọi là nửa Euler.

Rõ ràng, mọi đồ thị Euler đều là nửa Euler nhưng điều ngược lại không đúng.

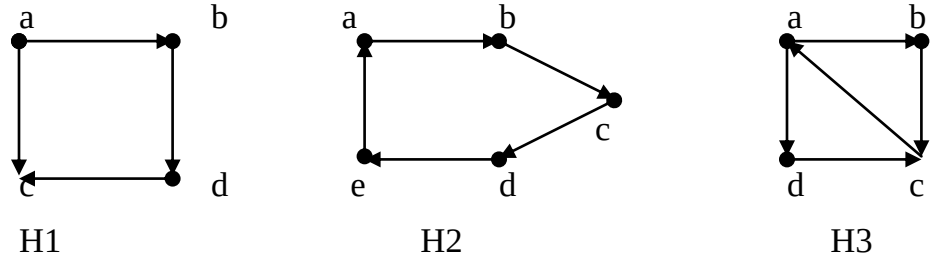
Ví dụ 1. Xét các đồ thị G1, G2, G3 trong Hình 4.1.



Hình 6.1. Đồ thị vô hướng G1, G2, G3.

Đồ thị G1 là đồ thị Euler vì nó có chu trình Euler a, e, c, d, e, b, a. Đồ thị G3 không có chu trình Euler nhưng chứa đường đi Euler a, c, d, e, b, d, a, b vì thế G3 là nửa Euler. G2 không có chu trình Euler cũng như đường đi Euler.

Ví dụ 2. Xét các đồ thị có hướng H1, H2, H3 trong Hình 4.2.



Hình 4.2. Đồ thị có hướng H1, H2, H3.

Đồ thị H2 là đồ thị Euler vì nó chứa chu trình Euler a, b, c, d, e, a vì vậy nó là đồ thị Euler. Đồ thị H3 không có chu trình Euler nhưng có đường đi Euler a, b, c, a, d, c nên nó là đồ thị nửa Euler. Đồ thị H1 không chứa chu trình Euler cũng như chu trình Euler.

4.2. Thuật toán tìm chu trình Euler

Để tìm một chu trình Euler của đồ thị ta sử dụng kết quả của định lý sau.

Định lý 1. *Điều kiện cần và đủ để đồ thị G=<V,E> là Euler***.** Đồ thị vô hướng liên thông G=<V, E> là đồ thị Euler khi và chỉ khi mọi đỉnh của G đều có bậc chẵn. Đồ thị có

hướng liên thông yếu G=<V, E> là đồ thị Euler khi và chỉ khi tất cả các đỉnh của nó đều có bán đỉnh bậc ra bằng bán đỉnh bậc vào (điều này làm cho đồ thị là liên thông mạnh).

4.2.1. Chứng minh đồ thị là Euler

Đối với đồ thị vô hướng, để chứng minh đồ thi có là Euler hay không ta chỉ cần thực hiện:

- Kiểm tra đồ thị có liên thông hay không? Điều này dễ dàng thực hiện bằng cách kiểm tra DFS(u) = V hoặc BFS(u) = V thì ta kết luận đồ thị là liên thông (u là đỉnh bất kỳ của đồ thị).
- Sử dụng tính chất của ma trận kề biểu đồ thị vô hướng để tính toán bậc của các đỉnh.

- Vì BFS(1) = { 1, 2, 6, 3, 5, 7, 4, 11, 8, 10, 12, 9, 13} = V. Do vậy, G liên thông.
- Ta lại có :

deg(1) = deg(13) = 2.
deg (2) = deg(3) = 4
deg(4) = deg(5) = 4
deg(6) = deg(7) = 4
deg(8) = deg(9) = 4
deg(10) = deg(11) = deg(12) = 4

Chú ý: Tổng các phần tử của hàng u (cột u) là bậc của đỉnh u. Ví dụ tổng các phần tử của hàng 1 là 2 nên deg(1) = 2.

0	1	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	1	0	0	0	1	1	0	0	1	0	0
0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	1	0	0	1	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	1	0	1	1
0	0	0	0	0	0	0	1	1	0	1	1	0
0	0	1	1	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	1	1	0	1
0	0	0	0	0	0	0	0	1	0	0	1	0

Đối với đồ thị có hướng, để chứng minh đồ thi có là Euler hay không ta chỉ cần thực hiện:

- Kiểm tra đồ thị có liên thông yếu hay không? Điều này dễ dàng thực hiện bằng cách kiểm tra nếu tồn tại đỉnh u$\in$ V để DFS(u) = V hoặc BFS(u) = V thì ta kết luận đồ thị là liên thông yếu.
- Sử dụng tính chất của ma trận kề biểu đồ thị có hướng để tính bán đỉnh bậc ra và bán đỉnh bậc vào của các đỉnh. Bán đỉnh bậc ra của đỉnh u là deg+(u) là số các số 1 của hàng u. Bán đỉnh bậc vào của đỉnh u là deg-(u) là số các số 1 của cột u.

Ví dụ để chứng minh đồ thị có hướng được biểu diễn dưới dạng ma trận kề như dưới đây ta thực hiện như sau:

+ Vì BFS(1) = { 1, 2, 3, 5, 4, 11, 6, 7, 10, 12, 8, 9, 13} = V. Do vậy, G liên thông yếu.
+ Ta lại có:

$deg^+(2)= deg^-(2)=deg^+(3)= eg^-(3) =2$
$deg^+(4)=deg^-(4)=deg^+(5)=deg^-(5) =2$
$deg^+(6)=deg^-(6)=deg^+(7)=deg^-(7) =2$
$deg^+(8)=deg^-(8)=deg^+(9)=deg^-(9) =2$
$deg^+(10) = deg^-(10) = 2$
$deg^+(11) = deg^-(11) =2$
$deg^+(12) = deg^-(12) =2$
$deg^+(1)=deg^-(1)=deg^-(13)=deg^+(13) =1$

0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	0

G liên thông yếu và có bán đỉnh bậc ra bằng bán đỉnh bậc vào nên G là đồ thị Euler.

4.2.2. Biểu diễn thuật toán tìm chu trình Euler

Để tìm một chu trình Euler trên đồ thị, ta thực hiện theo thuật toán trong Hình 4.3 dưới đây:

Thuật toán Euler-Cycle(u):

Bước 1 (Khởi tạo) :

```
stack = ∅ ;    //Khởi tạo một stack bắt đầu là ∅
CE = ∅ ;       //Khởi tạo mảng CE bắt đầu là ∅
Push (stack, u) ; //Đưa đỉnh u vào ngăn xếp
```

Bước 2 (Lặp):

```
while (stack≠∅ ) do { //Lặp cho đến khi stack rỗng
    s = get(stack); //Lấy đỉnh ở đầu ngăn xếp
    if ( Ke(s) ≠ ∅ ) then { // Nếu danh sách Ke(s) chưa rỗng
        t =< Đỉnh đầu tiên trong Ke(s)>;
        Push(stack, t)' //Đưa t vào stack;
        E = E \ (s,t); // Loại bỏ cạnh (s,t);
    }
    else { //Trường hợp Ke(s)=∅
        s = Pop(stack);// Đưa s ra khỏi ngăn xếp
        s ⇒CE; //Đưa s sang CE
    }
}
```

Bước 3 (Trả lại kết quả) :

```
<Lật ngược lại các đỉnh trong CE ta được chu trình Euler> ;
```

Hình 4.3. Thuật toán tìm chu trình Euler bắt đầu tại đỉnh u

4.2.3. Kiểm nghiệm thuật toán

Ví dụ ta cần tìm một chu trình Euler bắt đầu tại đỉnh u=1 trên đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề như dưới đây. Khi đó, các bước thực hiện của thuật toán được thực hiện như Bảng 4.1 (chú ý phần chứng minh ta đã thực hiện ở trên).

Bước	Trạng thái Stack	Giá trị CE
1	1	ϕ
2	1, 2	ϕ
3	1, 2, 3	ϕ
4	1, 2, 3, 4	ϕ
5	1, 2, 3, 4,7	ϕ
6	1, 2, 3, 4,7,5	ϕ
7	1, 2, 3, 4,7,5,2	ϕ
8	1, 2, 3, 4,7,5,2,6	ϕ
9	1, 2, 3, 4,7,5,2,6,1	ϕ
10	1, 2, 3, 4,7,5,2,6	1
11	1, 2, 3, 4,7,5,2,6,5	1
12	1, 2, 3, 4,7,5,2,6,5,3	1
13	1, 2, 3, 4,7,5,2,6,5,3,11	1
14	1, 2, 3, 4,7,5,2,6,5,3,11,4	1
15	1, 2, 3, 4,7,5,2,6,5,3,11,4,8	1
16	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,7	1
17	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,7,6	1
18	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,7	1,6
19	1, 2, 3, 4,7,5,2,6,5,3,11,4,8	1,6,7
20	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9	1,6,7
21	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10	1,6,7
22	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,8	1,6,7
23	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10	1,6,7,8
24	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11	1,6,7,8
25	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12	1,6,7,8
26	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9	1,6,7,8
27	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9,13	1,6,7,8
28	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9,13,12	1,6,7,8
29	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9,13,12,10	1,6,7,8
Đưa lần lượt các đỉnh trong Stack sang CE cho đến khi stack=∅		
30 -..	CE = 1, 6, 7, 8, 10, 12, 13, 9, 12, 11, 10, 9, 8, 4, 11, 3, 5, 6, 2, 5, 7, 4, 3, 2, 1	
Lật ngược lại các đỉnh trong CE ta được chu trình Euler		
1- 2- 3- 4- 7-5-2-6-5-3-11-4-8-9-10-11-12-9-13-12-10-8-7-6-1		

4.2.4. Cài đặt thuật toán

Chương trình tìm một chu trình Euler của đồ thị bắt đầu tạo đỉnh u trên đồ thị vô hướng liên thông được cài đặt theo khuôn dạng đồ thị biểu diễn dưới dạng ma trận kề. Các thủ tục chính bao gồm:

- Thủ tục Init() : đọc dữ liệu theo khuôn dạng biểu diễn ma trận kề.
- Thủ tục Kiemtra(): Kiểm tra xem G có là Euler hay không.
- Thủ tục Euler-Cycle (u) : Xây dựng chu trình Euler bắt đầu tại đỉnh u.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#define MAX 50
#define     TRUE 1
#define FALSE     0
int A[MAX][MAX], n, u=1;
void Init(void){
        int i, j;FILE *fp;
        fp = fopen("CTEULER.IN", "r");
        fscanf(fp,"%d", &n);
        printf("\n So dinh do thi:%d",n);
        printf("\n Ma tran ke:");
        for(i=1; i<=n;i++){
                printf("\n");
                for(j=1; j<=n;j++){
                        fscanf(fp,"%d", &A[i][j]);
                        printf("%3d", A[i][j]);
                }
        }
        fclose(fp);
}
int Kiemtra(void){
        int i, j, s, d;    d=0;
        for(i=1; i<=n;i++){
                s=0;
                for(j=1; j<=n;j++)
                        s+=A[i][j];
                if(s%2) d++;
        }
        if(d>0) return(FALSE);
        return(TRUE);
}
void Euler-Cycle( int u){
```

```
        int v, x, top, dCE;
        int stack[MAX], CE[MAX];
        top=1; stack[top]=u;dCE=0;
        do {
                v = stack[top];x=1;
                while (x<=n && A[v][x]==0)
                        x++;
                if (x>n) {
                        dCE++; CE[dCE]=v; top--;
                }
                else {
                        top++; stack[top]=x;
                        A[v][x]=0; A[x][v]=0;
                }
        } while(top!=0);
        printf("\n Co chu trinh Euler:");
        for(x=dCE; x>0; x--)
                printf("%3d", CE[x]);
}
void main(void){
        clrscr(); Init();
        if(Kiemtra())
                Tim();
        else printf("\n Khong co chu trinh Euler");
}
```

4.3. Thuật toán tìm đường đi Euler

Một đồ thị không có chu trình Euler nhưng vẫn có thể có đường đi Euler. Để tìm một đường đi Euler trên đồ thị vô hướng ta sử dụng kết quả của định lý 2. Để tìm một đường đi Euler trên đồ thị có hướng ta sử dụng kết quả của định lý 3.

Định lý 2. Đồ thị vô hướng liên thông G =<V,E> là đồ thị nửa Euler khi và chỉ khi G có 0 hoặc 2 đỉnh bậc lẻ. Trong trường hợp G có hai đỉnh bậc lẻ, đường đi Euler xuất phát tại một đỉnh bậc lẻ và kết thúc tại đỉnh bậc lẻ còn lại. Trong trường hợp G có 0 đỉnh bậc lẻ G chính là đồ thị Euler.

Định lý 3. Đồ thị có hướng liên thông yếu G =<V,E> là đồ thị nửa Euler khi và chỉ khi tồn tại đúng hai đỉnh u, v $\in$ V sao cho $deg^+(u) - deg^-(u) = deg^-(v) - deg^+(v)=1$, các đỉnh s$\neq$ u, s$\neq$ v còn lại có $deg^+(s) = deg^-(s)$. Đường đi Euler sẽ xuất phát tại đỉnh u và kết thúc tại đỉnh v.

4.3.1. Chứng minh đồ thị là nửa Euler

Để chứng tỏ đồ thị vô hướng G =<V,E> là nửa Euler ta cần thực hiện:

- Chứng tỏ đồ thị đã cho liên thông. Điều này dễ ràng thực hiện được bằng cách sử dụng hai thủ tục DFS(u) hoặc BFS(u).
- Có 0 hoặc hai đỉnh bậc lẻ. Sử dụng tính chất của các phương pháp biểu diễn đồ thị để tìm ra bậc của mỗi đỉnh.

Ví dụ. Chứng minh rằng, đồ thị vô hướng liên thông G =<V,E> được biểu diễn dưới dạng ma trận kề dưới đây là đồ thị nửa Euler.

Chứng minh. Theo tính chất của ma trận kề, tổng các phần tử hàng u là bậc của đỉnh u. Vì vậy ta có:

deg(1) = deg(13) = 3
deg (2) = deg(3) = deg(11) = 4
deg(12) = deg(6) = deg(7) = 4
deg(8) = deg(9) = 4
deg(5) = deg(4) = deg(10) = 6

G liên thông và có 2 đỉnh bậc lẻ u=1 và u=13 nên G là nửa Euler.

0	1	0	0	1	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	1	0	1	0	1	1	0	1	1	0	0
1	1	1	1	0	1	1	0	0	0	0	0	0
1	1	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	1	0	0	1	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	1	0	1	1
0	0	0	1	0	0	0	1	1	0	1	1	1
0	0	1	1	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	1	1	0	1
0	0	0	0	0	0	0	0	1	1	0	1	0

Để chứng tỏ đồ thị có hướng G =<V,E> là nửa Euler ta cần thực hiện:

- Chứng tỏ đồ thị đã cho liên thông yếu. Điều này dễ ràng thực hiện được bằng cách sử dụng hai thủ tục DFS(u) hoặc BFS(u).
- Có hai đỉnh u và v thỏa mãn $deg^+(u) - deg^-(u) = deg^-(v) - deg^+(v) = 1$.
- Các đỉnh $s \neq u$, $s \neq v$ còn lại có $deg^+(s) = deg^-(s)$.

Ví dụ. Chứng minh rằng, đồ thị có hướng liên thông yếu G =<V,E> được biểu diễn dưới dạng ma trận kề dưới đây là đồ thị nửa Euler?

Chứng minh. Theo tính chất của ma trận kề, $deg^+(u)$ là tổng các phần tử hàng u, $deg^-(u)$ là tổng các phần tử cột u. Vì vậy ta có:

$deg^+(2) = deg^-(2) = deg^+(3) = deg^-(3) = 2$
$deg^+(6) = deg^-(6) = deg^+(7) = deg^-(7) = 2$
$deg^+(8) = deg^-(8) = deg^+(9) = deg^-(9) = 2$
$deg^+(11) = deg^-(11) = deg^+(12) = deg^-(12) = 2$
$deg^+(5) = deg^-(5) = deg^+(4) = deg^-(4) = deg^+(10) = deg^-(10) = 3$
$deg^+(1) - deg^-(1) = deg^-(13) - deg^+(13) = 1$

G liên thông yếu và có 2 đỉnh u=1 và u=13 thỏa mãn điều kiện nên G là nửa Euler.

0	1	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	1	1	0	0
0	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	0

4.3.2. Thuật toán tìm đường đi Euler

Thuật toán tìm đường đi Euler và chu trình Euler chỉ khác nhau duy nhất ở một điểm đó là đầu vào của thuật toán. Đối với thuật toán tìm chu trình Euler, đầu vào thuật toán là đỉnh u∈V bất kỳ. Đối với thuật toán tìm đường đi trình Euler, đầu vào thuật toán là đỉnh u∈V là đỉnh bậc lẻ đầu tiên trong trường hợp đồ thị vô hướng. Đối với đồ thị có hướng, đỉnh u∈V là đỉnh có $deg^+(u)-deg^-(u)=1$. Thuật toán tìm đường đi Euler trên đồ thị vô hướng hoặc có hướng được mô tả chi tiết trong Hình 4.4.

Thuật toán Euler-Path (u):

- u là đỉnh bậc lẻ đầu tiên nếu G là đồ thị vô hướng
- u là đỉnh có $deg^+(u) - deg^-(u) =1$.

Bước 1 (Khởi tạo) :

```
stack = ∅ ;    //Khởi tạo một stack bắt đầu là ∅
dCE = ∅ ;      //Khởi tạo mảng dCE bắt đầu là ∅
Push (stack, u) ; //Đưa đỉnh u vào ngăn xếp
```

Bước 2 (Lặp):

```
while (stack≠∅ ) do { //Lặp cho đến khi stack rỗng
        s = get(stack); //Lấy đỉnh ở đầu ngăn xếp
        if ( Ke(s) ≠ ∅ ) then { // Nếu danh sách Ke(s) chưa rỗng
                t =< Đỉnh đầu tiên trong Ke(s)>;
                Push(stack, t)' //Đưa t vào stack;
                E = E \ (s,t); // Loại bỏ cạnh (s,t);
        }
        else { //Trường hợp Ke(s)=∅
                s = Pop(stack);// Đưa s ra khỏi ngăn xếp
                s ⇒dCE; //Đưa s sang dCE
        }
}
```

Bước 3 (Trả lại kết quả) :

```
<Lật ngược lại các đỉnh trong dCE ta được đường đi Euler> ;
```

Hình 4.4. Thuật toán tìm đường đi Euler trên đồ thị.

4.3.3. Kiểm nghiệm thuật toán

Ví dụ ta cần tìm đường đi Euler trên đồ thị có hướng liên thông yếu được biểu diễn dưới dạng ma trận kề trong Mục 4.3.1. Khi đó, đỉnh u có deg+(u)-deg-(u)=1 là đỉnh 1. Kết quả thực hiện của thuật toán Hình 4.4 được thể hiện trong Bảng 4.2 dưới đây.

Bước	Trạng thái Stack	Giá trị dCE
1	1	ϕ
2	1, 2	
3	1, 2, 3	
4	1, 2, 3, 4	
5	1, 2, 3, 4,7	
6	1, 2, 3, 4,7,5	
7	1, 2, 3, 4,7,5,3	
8	1, 2, 3, 4,7,5,3,11	
9	1, 2, 3, 4,7,5,3,11,10	
10	1, 2, 3, 4,7,5,3,11,10,8	
11	1, 2, 3, 4,7,5,3,11,10,8,4	
12	1, 2, 3, 4,7,5,3,11,10,8,4,10	
13	1, 2, 3, 4,7,5,3,11,10,8,4,10,12	
14	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9	
15	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8	
16	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7	
17	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6	
18	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1	
19	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5	
20	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4	
21	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11	
22	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12	
23	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13	
24	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9	
25	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9,10	
26	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9,10,13	
27	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9,10	13,
28	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9	13,10
29	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13	13,10,9
30	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12	13,10,9,13
31	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11	13,10,9,13,12
32	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4	13,10,9,13,12,11
33	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5	13,10,9,13,12,11,4
34	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,6	13,10,9,13,12,11,4
35	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,6,2	13,10,9,13,12,11,4
36	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,6,2,5	13,10,9,13,12,11,4
Đưa lần lượt các đỉnh trong Stack sang dCE		
37 ...	dCE = 13,10,9,13,12,11,4, 5, 2, 6, 5, 1, 6, 7, 8, 9, 12, 10, 4, 8, 10, 11, 3, 5, 7, 4, 3, 2, 1	
Lật ngược lại các đỉnh trong CE ta được đường đi Euler		
1- 2- 3- 4- 7- 5- 3- 11- 10- 8 -4- 10- 12- 9- 8- 7- 6- 1-5- 6- 2- 5- 4-11- 12- 13-9-10-13		

4.3.4. Cài đặt thuật toán

Chương trình tìm một đường đi Euler của đồ thị bắt đầu tạo đỉnh u trên đồ thị vô hướng liên thông được cài đặt theo khuôn dạng đồ thị biểu diễn dưới dạng ma trận kề. Các thủ tục chính bao gồm:

- Thủ tục Init() : đọc dữ liệu theo khuôn dạng biểu diễn ma trận kề.
- Thủ tục Kiemtra(): Kiểm tra xem G có là nửa Euler hay không.
- Thủ tục Euler-Cycle (u) : Xây dựng đường đi Euler bắt đầu tại đỉnh u (đỉnh bậc lẻ đầu tiên).

Chương trình tìm đường đi Euler được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define    TRUE 1
#define FALSE   0
void Init(int A[][MAX], int *n){
    int i, j;FILE *fp;
    fp = fopen("DDEULER.IN", "r");
    fscanf(fp,"%d", n);
    printf("\n So dinh do thi:%d",*n);
    printf("\n Ma tran ke:");
    for(i=1; i<=*n;i++){
            printf("\n");
            for(j=1; j<=*n;j++){
                    fscanf(fp,"%d", &A[i][j]);
                    printf("%3d", A[i][j]);
            }
    }
    fclose(fp);
}
int Kiemtra(int A[][MAX], int n, int *u){
    int i, j, s, d;
    d=0;
    for(i=1; i<=n;i++){
            s=0;
            for(j=1; j<=n;j++)
                    s+=A[i][j];
            if(s%2){
```

```
                d++;*u=i;
            }
        }
        if(d!=2) return(FALSE);
        return(TRUE);
    }
    void DDEULER(int A[][MAX], int n, int u){
        int v, x, top, dCE;
        int stack[MAX], CE[MAX];
        top=1; stack[top]=u;dCE=0;
        do {
            v = stack[top];x=1;
            while (x<=n && A[v][x]==0)
                x++;
            if (x>n) {
                dCE++; CE[dCE]=v; top--;
            }
            else {
                top++; stack[top]=x;
                A[v][x]=0; A[x][v]=0;
            }
        } while(top!=0);
        printf("\n Co duong di Euler:");
        for(x=dCE; x>0; x--)
            printf("%3d", CE[x]);
    }
    void main(void){
        int A[MAX][MAX], n, u;
        clrscr(); Init(A, &n);
        if(Kiemtra(A,n,&u))
            DDEULER(A,n,u);
        else printf("\n Khong co duong di Euler");
        getch();
    }
```

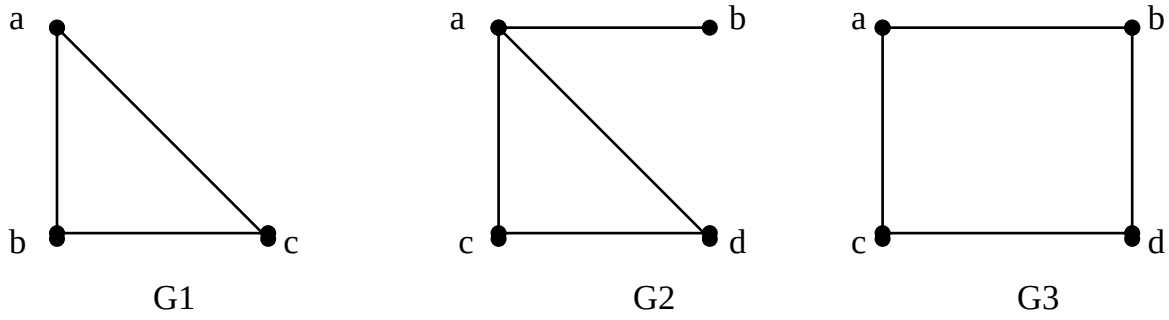
4.4. Đồ thị Hamilton

Với đồ thị Euler, chúng ta quan tâm tới việc duyệt các cạnh của đồ thị mỗi cạnh đúng một lần, thì trong mục này, chúng ta xét đến một bài toán tương tự nhưng chỉ khác nhau là ta chỉ quan tâm tới các đỉnh của đồ thị, mỗi đỉnh đúng một lần. Sự thay đổi này tưởng như không đáng kể, nhưng thực tế có nhiều sự khác biệt trong khi giải quyết bài toán.

Định nghĩa. Đường đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần được gọi là đường đi Hamilton. Chu trình bắt đầu tại một đỉnh *v* nào đó qua tất cả các đỉnh còn lại mỗi đỉnh đúng một lần sau đó quay trở lại *v* được gọi là chu trình Hamilton. Đồ thị có chu trình Hamilton được gọi là đồ thị Hamilton. Đồ thị có đường đi Hamilton được gọi là đồ thị nửa Hamilton.

Như vậy, một đồ thị Hamilton bao giờ cũng là đồ thị nửa Hamilton nhưng điều ngược lại không luôn luôn đúng. Ví dụ sau sẽ minh họa cho nhận xét này.

Ví dụ. Đồ thị đồ thi hamilton *G3*, nửa Hamilton *G2* và *G1*.



Hình 4.5. Đồ thị đồ thi hamilton G3, nửa Hamilton G2 và G1.

4.4.1. Thuật toán tìm tất cả các chu trình Hamilton

Cho đến nay, việc tìm ra một tiêu chuẩn để nhận biết đồ thị Hamilton vẫn còn mở, mặc dù đây là vấn đề trung tâm của lý thuyết đồ thị. Cho đến nay cũng vẫn chưa có thuật toán hiệu quả để kiểm tra một đồ thị có phải là đồ thị Hamilton hay không. Hình 4.6 dưới đây mô tả thuật toán liệt kê tất cả chu trình Hamilton bắt đầu tại đỉnh k.

```
Thuật toán Hamilton( int k) {
/* Liệt kê các chu trình Hamilton của đồ thị bằng cách phát triển dãy đỉnh
(X[1], X[2], . . ., X[k-1] ) của đồ thị G = (V, E) */
        for y∈ Ke(X[k-1]) {
                if (k==n+1) and (y == v0) then
                    Ghinhan(X[1], X[2], . . ., X[n], v0);
                else {
                        X[k]=y; chuaxet[y] = false;
                        Hamilton(k+1);
                        chuaxet[y] = true;
                }
        }
}
```

Hình 4.6. Thuật toán liệt kê các chu trình Hamilton bắt đầu tại đỉnh k.

Khi đó, việc liệt kê chu trình Hamilton được thực hiện như sau:

Begin

*for (v∈V) chuaxet[v] = true; /*thiết lập trạng thái các đỉnh*/*

*X[1] = v0; (*v0 là một đỉnh nào đó của đồ thị*)*

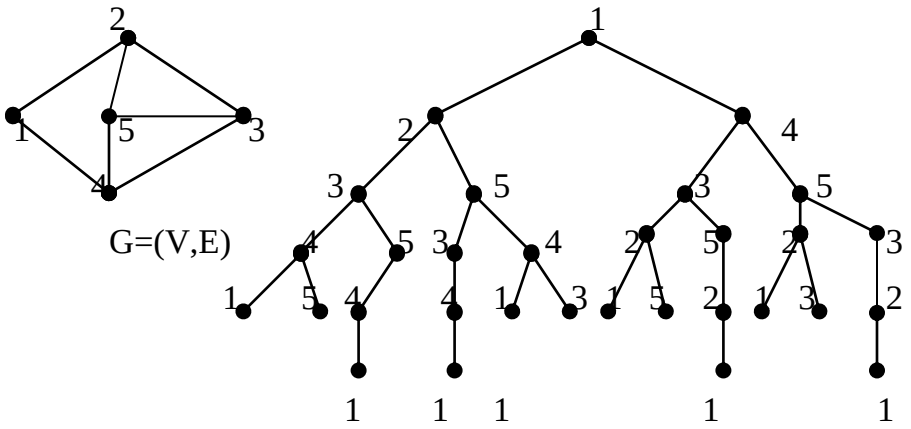
chuaxet[v0] = false;

Hamilton(2);

End.

4.4.2. Kiểm nghiệm thuật toán

Ví dụ với đồ thị G=<V, E> dưới đây sẽ cho ta cây tìm kiếm chu trình Hamilton thể hiện thuật toán trên được mô tả như trong Hình 4.6.



Hình 4.7. Cây tìm kiếm chu trình Hamilton.

4.4.3. Cài đặt thuật toán

Chương trình liệt kê các chu trình Hamilton được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define     TRUE 1
#define FALSE    0
int A[MAX][MAX], C[MAX], B[MAX];
int n,i, d;
void Init(void){
    int i, j;FILE *fp;
```

```
        fp= fopen("CCHMTON.IN", "r");
        if(fp==NULL){
                printf("\n Khong co file input");
                getch(); return;
        }
        fscanf(fp,"%d",&n);
        printf("\n So dinh do thi:%d", n);
        printf("\n Ma tran ke:");
        for(i=1; i<=n; i++){
                printf("\n");
                for(j=1; j<=n; j++){
                        fscanf(fp, "%d", &A[i][j]);
                        printf("%3d", A[i][j]);
                }
        }
        fclose(fp);
        for (i=1; i<=n;i++)
                C[i]=0;
}
void Result(void){
        int i;
        printf("\n ");
        for(i=n; i>=0; i--)
                printf("%3d", B[i]);
        d++;
}
void Hamilton(int *B, int *C, int i){
        int j, k;
        for(j=1; j<=n; j++){
                if(A[B[i-1]][j]==1 && C[j]==0){
                        B[i]=j; C[j]=1;
                        if(i<n) Hamilton(B, C, i+1);
                        else if(B[i]==B[0]) Result();
                        C[j]=0;
                }
        }
}
void main(void){
        B[0]=1; i=1;d=0;    Init();
        Hamilton(B,C,i);
        if(d==0)                printf("\n Khong co chu trinh Hamilton");
        }
```

4.4.3. Cài đặt thuật toán

Cũng giống như thuật toán tìm chu trình Hamilton, thuật toán tìm đường đi Hamilton được cài đặt như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define     TRUE 1
#define FALSE    0
int A[MAX][MAX], C[MAX], B[MAX];
int n,i, d;
void Init(void){
    int i, j;FILE *fp;
    fp= fopen("DDHMTON.IN", "r");
    if(fp==NULL){
        printf("\n Khong co file input");
        getch(); return;
    }
    fscanf(fp,"%d",&n);
    printf("\n So dinh do thi:%d", n);
    printf("\n Ma tran ke:");
    for(i=1; i<=n; i++){
        printf("\n");
        for(j=1; j<=n; j++){
            fscanf(fp, "%d", &A[i][j]);
            printf("%3d", A[i][j]);
        }
    }
    fclose(fp);
    for (i=1; i<=n;i++)
        C[i]=0;
}
void Result(void){
    int i;
    printf("\n ");
    for(i=n; i>0; i--)
        printf("%3d", B[i]);
    d++;
}
void Hamilton(int *B, int *C, int i){
```

```
        int j, k;
        for(j=1; j<=n; j++){
                if(A[B[i-1]][j]==1 && C[j]==0){
                        B[i]=j; C[j]=1;
                        if(i<n) Hamilton(B, C, i+1);
                        else  Result();
                        C[j]=0;
                }
        }
}
void main(void){
        B[0]=1; i=1;d=0;
   Init();
        Hamilton(B,C,i);
        if(d==0)
                printf("\n Khong co duong di Hamilton");
        getch();
}
```

4.5. Những điểm cần ghi nhớ

- ✓ Khái niệm và định nghĩa về đồ thị Euler, đồ thị nửa Euler, đồ thị Hamilton, đồ thị nửa Hamilton.
- ✓ Nắm vững và phân biệt rõ sự khác biệt giữa chu trình (đường đi) Euler và chu trình (đường đi Hamilton).
- ✓ Phương pháp hiểu rõ bản chất của thuật toán là cài đặt và kiểm chứng thuật toán bằng cách viết chương trình.

BÀI TẬP

1. Cho đồ thị vô hướng liên thông G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là Euler?

b) Xây dựng thuật toán tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u∈V?

c) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

d) Tìm một chu trình Euler bắt đầu tại đỉnh u=5? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

e) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1	1	0	0
0	1	0	0	0	0	0	0	1	1	0	0	1
1	0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	1	0	1	0	1
0	1	0	1	0	0	0	0	0	0	1	1	0
0	0	1	0	1	0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	1	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	1	0	1	0	0

2. Cho đồ thị vô hướng liên thông G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là nửa Euler?

b) Xây dựng thuật toán tìm một đường đi Euler của đồ thị?

c) Tìm một đường đi Euler của đồ thị? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

d) Viết chương trình tìm một đường đi Euler của đồ thị?

0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1	1	0	0
0	1	0	0	0	0	0	0	1	1	0	0	1
1	0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	0	0	1	0	1
0	1	0	1	0	0	0	0	0	0	1	1	0
0	0	1	0	1	0	0	0	0	0	0	0	1
0	1	1	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	1	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	1	0	1	0	0

3. Cho đồ thị có hướng liên thông yếu G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là Euler?

b) Xây dựng thuật toán tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u∈V?

c) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

d) Tìm một chu trình Euler bắt đầu tại đỉnh u=5? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

e) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

4. Cho đồ thị có hướng liên thông yếu G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là nửa Euler?

b) Xây dựng thuật toán tìm một đường đi của đồ thị?

c) Tìm một đường đi Euler của đồ thị? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

e) Viết chương trình tìm một đường đi duler của đồ thị?

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

5. Cho đồ thị vô hướng liên thông được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 4, 6 }.	Ke(5) = { 7, 9 }.	Ke(9) = { 3, 5, 7, 13 }.
Ke(2) = { 3, 8, 10, 11}.	Ke(6) = { 1, 4, 10, 12 }.	Ke(10) = { 2, 3, 6, 12 }.
Ke(3) = { 2, 9, 10, 13 }.	Ke(7) = { 5, 9, 11, 13 }.	Ke(11) = { 2, 7, 8, 13 }.
Ke(4) = { 1, 6, 8, 12 }.	Ke(8) = { 2, 4, 11, 12 }.	Ke(12) = { 4, 6, 8, 10 }.
		Ke(13) = { 3, 7, 9, 11 }.

Hãy thực hiện:

a) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

b) Tìm một chu trình Euler bắt đầu tại đỉnh u=5? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

6. Cho đồ thị vô hướng liên thông được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 4, 6 }.	Ke(5) = { 7, 9 }.	Ke(9) = { 3, 5, 7, 13 }.
Ke(2) = { 3, 8, 10, 11}.	Ke(6) = { 1, 10, 12 }.	Ke(10) = { 2, 3, 6, 12 }.
Ke(3) = { 2, 9, 10, 13 }.	Ke(7) = { 5, 9, 11, 13 }.	Ke(11) = { 2, 7, 8, 13 }.

Ke(4) = { 1, 8, 12 }. Ke(8) = { 2, 4, 11, 12 }. Ke(12) = { 4, 6, 8, 10 }.
Ke(13) = { 3, 7, 9, 11 }.

Hãy thực hiện:

a) Xây dựng thuật toán tìm một đường đi Euler của đồ thị?

b) Tìm một đường đi Euler của đồ thị? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình tìm một đường đi của đồ thị bắt đầu tại đỉnh u?

7. Cho đồ thị có hướng liên thông yếu được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 6 }.	Ke(5) = { 7 }.	Ke(9) = { 5, 7 }.
Ke(2) = { 3, 8}.	Ke(6) = { 10, 12 }.	Ke(10) = { 2, 3 }.
Ke(3) = { 9, 13 }.	Ke(7) = { 11, 13 }.	Ke(11) = { 2, 8 }.
Ke(4) = { 1, 6 }.	Ke(8) = { 4, 12 }.	Ke(12) = { 4, 10 }.
		Ke(13) = { 9, 11 }.

Hãy thực hiện:

a) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

b) Tìm một chu trình Euler bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

8. Cho đồ thị có hướng liên thông yếu được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 6 }.	Ke(5) = { 7 }.	Ke(9) = { 5, 7 }.
Ke(2) = { 3, 8}.	Ke(6) = { 10, 12 }.	Ke(10) = { 2, 3 }.
Ke(3) = { 9, 13 }.	Ke(7) = { 11, 13 }.	Ke(11) = { 2, 8 }.
Ke(4) = { 1 }.	Ke(8) = { 4, 12 }.	Ke(12) = { 4, 10 }.
		Ke(13) = { 9, 11 }.

Hãy thực hiện:

a) Trình bày thuật toán tìm một đường đi Euler trên đồ thị có hướng?

b) Tìm một đường đi Euler của đồ thị?

c) Viết chương trình tìm một đường đi Euler của đồ thị?

CHƯƠNG 5. CÂY KHUNG CỦA ĐỒ THỊ

Nội dung chính của chương này đề cập đến một loại đồ thị đơn giản nhất đó là cây. Cây được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau của tin học như tổ chức các thư mục, lưu trữ dữ liệu, biểu diễn tính toán, biểu diễn quyết định và tổ chức truyền tin. Những nội dung được trình bày bao gồm:

- ✓ Cây và các tính chất cơ bản của cây.
- ✓ Cây khung của đồ thị & các thuật toán cơ bản xây dựng cây khung của đồ thị.
- ✓ Bài toán tìm cây khung nhỏ nhất & các thuật toán tìm cây khung nhỏ nhất.
- ✓ Thuật toán Kruskal tìm cây bao trùm nhỏ nhất.
- ✓ Thuật toán Prim tìm cây bao trùm nhỏ nhất.

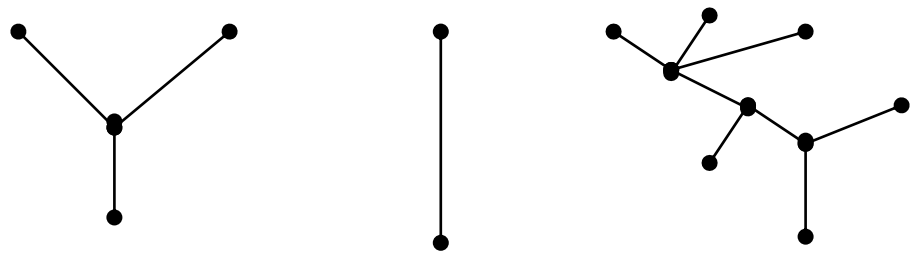
Bạn đọc có thể tìm thấy những chứng minh cụ thể cho các định lý, tính đúng đắn và độ phức tạp các thuật toán thông qua các tài liệu [1], [2].

5.1. Cây và một số tính chất cơ bản

Định nghĩa 1. Ta gọi cây là đồ thị vô hướng liên thông không có chu trình. Đồ thị không liên thông được gọi là rừng.

Như vậy, rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Ví dụ. Rừng gồm 3 cây trong hình 7.1.



Hình 5.1 . *Rừng gồm 3 cây T1, T2, T3.*

Cây được coi là dạng đồ thị đơn giản nhất của đồ thị. Định lý sau đây cho ta một số tính chất của cây.

Định lý. Giả sử T= <V, E> là đồ thị vô hướng n đỉnh. Khi đó những khẳng định sau là tương đương

a) T là một cây;

b) T không có chu trình và có n-1 cạnh;

c) T liên thông và có đúng n-1 cạnh;

d) T liên thông và mỗi cạnh của nó đều là cầu;

e) Giữa hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn;

f) T không chứa chu trình nhưng hễ cứ thêm vào nó một cạnh ta thu được đúng một chu trình;

Chứng minh. Định lý được chứng minh định lý thông qua các bước (a) =>(b) =>(c) => (d) =>(e) => (f) => (a). Những bước cụ thể của quá trình chứng minh bạn đọc có thể tìm thấy trong các tài liệu [1], [2].

Định nghĩa 2. Cho G là đồ thị vô hướng liên thông. Ta gọi đồ thị con T của G là một cây khung của G (Cây bao trùm) nếu T thoả mãn hai điều kiện:

a) T là một cây;

b) Tập đỉnh của T bằng tập đỉnh của G.

Trong lý thuyết đồ thị, người ta qua tâm đến hai bài toán cơ bản về cây:

Bài toán 1. Cho đồ thị vô hướng G =<V,E>. Hãy xây dựng một cây khung của đồ thị bắt đầu tại đỉnh u∈V.

Bài toán 2. Cho đồ thị vô hướng G =<V,E> có trọng số. Hãy xây dựng cây khung có độ dài nhỏ nhất.

Bài toán 1 được giải quyết bằng các thuật toán tìm kiếm cơ bàn: thuật toán DFS hoặc BFS. Bài toán 2 được giải quyết bằng thuật toán Kruskal hoặc PRIM.

5.2. Xây dựng cây khung của đồ thị dựa vào thuật toán DFS

Để tìm một cây khung trên đồ thị vô hướng liên thông ta có thể sử dụng kỹ thuật tìm kiếm theo chiều sâu. Giả sử ta cần xây dựng một cây bao trùm xuất phát tại đỉnh *u* nào đó. Trong cả hai trường hợp, mỗi khi ta đến được đỉnh *v* tức (*chuaxet[v] = False*) từ đỉnh u thì cạnh (*u,v*) được kết nạp vào cây khung. Kỹ thuật xây dựng cây khung bắt đầu tại đỉnh u dựa vào thuật toán DFS được mô tả trong Hình 5.2.

5.2.1. Mô tả thuật toán

```
Thuật toán Tree-DFS(u) {
        chuaxet[u] = False; //Bật trạng thái đỉnh u từ True trở thành False
        for v∈Ke(u) do { //Duyệt trên danh sách kề của đỉnh u
                if (chuaxet[v]) { //Nếu đỉnh v chưa được xét đến
                        T = T∪(u,v); //Hợp cạnh (u,v) vào cây khung
                        DFS(v); //Duyệt theo chiều sâu bắt đầu tại đỉnh v
                }
        }
}
```

Hình 5.2. Thuật toán Tree-DFS(u).

Khi đó, quá trình xây dựng cây khung bắt đầu tại đỉnh u được thực hiện như thuật toán trong Hình 5.3.

```
Thuật toán Tree-Graph-DFS() {
        for each u∈V do //Khởi tạo các đỉnh chưa xét
                chuaxet[u]= True;
        endfor;
        roof = <Đỉnh bất kỳ của đồ thị>; //Lấy một đỉnh bất kỳ làm gốc
        T = ∅;          //Thiết lập tập cạnh ban đầu của cây là ∅
        Tree-DFS(roof); //Thực hiện thuật toán Tree-DFS(roof)
        if (|T| <n-1) <Đồ thị không liên thông>;
        else <Ghi nhận tập cạnh Tcủa cây khung>
}
```

Hình 5.3. Thuật toán xây dựng cây khung dựa vào DFS.

5.2.2. Kiểm nghiệm thuật toán

Giả sử ta cần kiểm nghiệm thuật toán Tree-Graph-DFS với đỉnh bắt đầu u=1 trên đồ thị được biểu diễn dưới dạng ma trận kề dưới đây. Khi đó các bước thực hiện của thuật toán được thể hiện trong Bảng 5.1.

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bảng 5.1. Kiểm nghiệm thuật toán Tree-Graph-DFS

Bước	Tree-DFS(u) =?	T =?
1	1	T = ∅
2	1, 2	T = T∪(1,2)
3	1, 2, 3	T = T∪(2,3)

4	1, 2, 3, 4	T = T∪(3, 4)
5	1, 2, 3, 4, 5	T = T∪(3, 5)
6	1, 2, 3, 4, 5, 6	T = T∪(5, 6)
7	1, 2, 3, 4, 5, 6, 7	T = T∪(6, 7)
8	1, 2, 3, 4, 5, 6, 7, 8	T = T∪(7, 8)
9	1, 2, 3, 4, 5, 6, 7, 8, 9	T = T∪(8, 9)
10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	T = T∪(9, 10)
11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	T = T∪(10, 11)
12	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	T = T∪(11, 12)
13	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	T = T∪(12, 13)
Kết luận T = {(1,2), (2,3), (3,4), (3,5), (5,6), (6,7), (7,8), (8,9), (9,10), (10,11), (11,12), (12,13)		

5.2.3. Cài đặt thuật toán

Thuật toán Tree-Graph-DFS được cài đặt đối với đồ thị được biểu diễn dưới dạng ma trận kề. Các thủ tục chính được cài đặt bao gồm:

- Thủ tục Init() : đọc dữ liệu và thiết lập giá trị của mảng chuaxet[].
- Thủ tục Tree-DFS (u) : thuật toán DFS bắt đầu tại đỉnh u.
- Thủ tục Result(): ghi nhận tập cạnh của cây khung.

Chương trình xây dựng một cây khung được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 50
#define    TRUE 1
#define FALSE  0
int CBT[MAX][2], n, A[MAX][MAX], chuaxet[MAX], sc, QUEUE[MAX];
void Init(void){
    int i, j;FILE *fp;
    fp= fopen("BAOTRUM1.IN", "r");
    if(fp==NULL){
            printf("\n Khong co file input");
            getch(); return;
    }
    fscanf(fp,"%d",&n);
```

```
    printf("\n So dinh do thi:%d", n);
    printf("\n Ma tran ke:");
    for(i=1; i<=n; i++){
        printf("\n");
        for(j=1; j<=n; j++){
            fscanf(fp, "%d", &A[i][j]);
            printf("%3d", A[i][j]);
        }
    }
    fclose(fp);
    for (i=1; i<=n;i++)
        chuaxet[i]=TRUE;
}
void TREE_DFS(int i){
    int j; chuaxet[i] = False;
    if(sc==n-1) return;
    for(j=1; j<=n; j++){
        if (chuaxet[j] && A[i][j]){  sc++;
            CBT[sc][1]=i; CBT[sc][2]=j;
            if(sc==n-1) return;
            STREE_DFS(j);
        }
    }
}
void Result(void){
    int i, j;
    for(i=1; i<=sc; i++){
        printf("\n Canh %d:", i);
        for(j=1; j<=2; j++)
            printf("%3d", CBT[i][j]);
    }
}
void main(void){
    int i; Init(); sc=0; i=1; /* xây dựng cây bao trùm tại đỉnh 1*/
    TREE_DFS(i);
    if (sc<n-1) printf("\n Đồ thị không liên thông");
    else Result();
}
```

5.3. Xây dựng cây khung của đồ thị dựa vào thuật toán BFS

Để tìm một cây khung trên đồ thị vô hướng liên thông ta có thể sử dụng kỹ thuật tìm kiếm theo chiều rộng. Giả sử ta cần xây dựng một cây bao trùm xuất phát tại đỉnh *u*

nào đó. Trong cả hai trường hợp, mỗi khi ta đến được đỉnh *v* tức (*chuaxet[v] = False*) từ đỉnh u thì cạnh (*u,v*) được kết nạp vào cây khung.

5.3.1. Cài đặt thuật toán

Thuật toán xây dựng cây khung của đồ thị được mô tả như Hình 5.4.

```
Thuật toán Tree-BFS(u):
Begin
        Bước 1 (Khởi tạo):
                T = ∅; //Tập cạnh cây khung ban đầu.
                Queue = ∅; //Thiết lập hàng đợi ban đầu;
                Push(Queue, u); //Đưa u vào hàng đợi;
                chuaxet[u] = False;//Bật trạng thái đã xét của đỉnh u
        Bước 2 (Lặp):
                while (Queue≠∅ ) do { //Lặp cho đến khi hàng đợi rỗng
                        s = Pop(Queue); Lấy s ra khỏi hàng đợi
                        for each t∈Ke(s) do { //Lặp trên danh sách Ke(s)
                                if (chuaxet[t] ) then { //Nếu đỉnh t chuaxet
                                        Push(Queue, t);// Đưa t vào hàng đợi
                                        T = T∪(s,t); //Kết nạp (s,t) vào cây khung
                                        chuaxet[t] = False; //Ghi nhận t đã xét
                                endif ;
                        endfor ;
                endwwhile ;
        Bước 3 (Trả lại kết quả) :
                if (| T | < n-1 ) <Đồ thị không liên thông> ;
                else <Ghi nhận tập cạnh T của cây khung" ;
end.
```

Hình 5.4. Thuật toán Tree-BFS(u).

5.3.2. Kiểm nghiệm thuật toán

Giả sử ta cần kiểm nghiệm thuật toán Tree- BFS với đỉnh bắt đầu u=1 trên đồ thị được biểu diễn dưới dạng ma trận kề dưới đây. Khi đó các bước thực hiện của thuật toán được thể hiện trong Bảng 5.2.

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bảng 5.2. Kiểm nghiệm thuật toán Tree-BFS

Bước	Trạng thái hàng đợi: Tree-BFS(u) =?	T =?
1	1	T = ∅
2	2, 3, 4	T = T∪{(1,2), (1,3), (1,4) }
3	3, 4	T=T∪∅
4	4, 5	T=T∪(3,5)
5	5	T=T∪∅
6	6, 7, 8, 9	T = T∪{(5,6), (5,7), (5,8), (5,9)}
7	7, 8, 9	T = T∪∅
8	8, 9	T = T∪∅
9	9	T = T∪∅
10	10	T = T∪(9,10)
11	11, 12, 13	T = T∪{(10,11), (10,12), (10,13)}
12	12, 13	T = T∪∅
13	13	T = T∪∅
14	∅	T = T∪∅
Kết luận T = {(1,2), (1,3), (1,4), (3,5), (5,6), (5,7), (5,8), (5,9), (9,10), (10,11), (10,12), (10,13)		

5.3.3. Cài đặt thuật toán

Thuật toán Tree-BFS được cài đặt đối với đồ thị được biểu diễn dưới dạng ma trận kề. Các thủ tục chính được cài đặt bao gồm:

- Thủ tục Init() : đọc dữ liệu và thiết lập giá trị của mảng chuaxet[].
- Thủ tục Tree-BFS (u) : thuật toán BFS bắt đầu tại đỉnh u.
- Thủ tục Result(): ghi nhận tập cạnh của cây khung.

Chương trình xây dựng một cây khung được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 50
#define    TRUE 1
#define FALSE   0
int CBT[MAX][2], n, A[MAX][MAX], chuaxet[MAX], sc, QUEUE[MAX];
void Init(void){
    int i, j;FILE *fp;
    fp= fopen("BAOTRUM1.IN", "r");
    if(fp==NULL){
            printf("\n Khong co file input");
            getch(); return;
    }
    fscanf(fp,"%d",&n);
    printf("\n So dinh do thi:%d", n);
    printf("\n Ma tran ke:");
    for(i=1; i<=n; i++){
            printf("\n");
            for(j=1; j<=n; j++){
                    fscanf(fp, "%d", &A[i][j]);
                    printf("%3d", A[i][j]);
            }
    }
    fclose(fp);
    for (i=1; i<=n;i++)
            chuaxet[i]=TRUE;
}
void TREE_BFS(int u){
    int dauQ, cuoiQ, v, p;
    dauQ=1; cuoiQ=1; QUEUE[dauQ]=u;chuaxet[u]=FALSE;
    while(dauQ<=cuoiQ){
            v= QUEUE[dauQ]; dauQ=dauQ+1;
            for(p=1; p<=n; p++){
                    if(chuaxet[p] && A[v][p]){
                            chuaxet[p]=FALSE; sc++;
                            CBT[sc][1]=v; CBT[sc][2]=p;
```

```
                        cuoiQ=cuoiQ+1;
                        QUEUE[cuoiQ]=p;
                        if(sc==n-1) return;
                    }
                }
            }
        }

    void Result(void){
        int i, j;
        for(i=1; i<=sc; i++){
                printf("\n Canh %d:", i);
                for(j=1; j<=2; j++)
                    printf("%3d", CBT[i][j]);
        }
    }
    void main(void){
        int i; Init(); sc=0; i=1; /* xây dựng cây bao trùm tại đỉnh 1*/
        TREE_BFS(i);
        if (sc<n-1) printf("\n Đồ thị không liên thông");
        else Result();
    }
```

5.4. Bài toán xây dựng cây khung có độ dài nhỏ nhất

Bài toán tìm cây khung nhỏ nhất là một trong những bài toán tối ưu trên đồ thị có ứng dụng trong nhiều lĩnh vực khác nhau của thực tế. Bài toán được phát biểu như dưới đây.

5.4.1. Đặt bài toán

Cho $G=<V, E>$ là đồ thị vô hướng liên thông với tập đỉnh $V = \{1, 2, . . ., n\}$ và tập cạnh E gồm m cạnh. Mỗi cạnh e của đồ thị được gán với một số không âm $c(e)$ được gọi là độ dài cạnh. Giả sử $H=<V, T>$ là một cây khung của đồ thị G. Ta gọi độ dài $c(H)$ của cây khung H là tổng độ dài các cạnh: $c(H) = \sum_{e \in T} c(e)$. Bài toán được đặt ra là, trong số các cây khung của đồ thị hãy tìm cây khung có độ dài nhỏ nhất của đồ thị.

Để minh họa cho những ứng dụng của bài toán này, chúng ta có thể tham khảo hai mô hình thực tế của bài toán.

Bài toán nối mạng máy tính. Một mạng máy tính gồm n máy tính được đánh số từ *1, 2, . . ., n*. Biết chi phí nối máy *i* với máy *j* là *c[i, j], i, j = 1, 2, . . ., n*. Hãy tìm cách nối mạng sao cho chi phí là nhỏ nhất.

Bài toán xây dựng hệ thống cable. Giả sử ta muốn xây dựng một hệ thống cable điện thoại nối *n* điểm của một mạng viễn thông sao cho điểm bất kỳ nào trong mạng đều

có đường truyền tin tới các điểm khác. Biết chi phí xây dựng hệ thống cable từ điểm *i* đến điểm *j* là *c[i,j]*. Hãy tìm cách xây dựng hệ thống mạng cable sao cho chi phí là nhỏ nhất.

Để giải bài toán cây khung nhỏ nhất, chúng ta có thể liệt kê toàn bộ cây khung và chọn trong số đó một cây nhỏ nhất. Phương án như vậy thực sự không khả thi vì số cây khung của đồ thị là rất lớn cỡ n^{n-2}, điều này không thể thực hiện được với đồ thị với số đỉnh cỡ chục.

Để tìm một cây khung ta có thể thực bằng hai thuật toán: Thuật toán Kruskal và thuật toán PRIM.

5.4.2. Thuật toán Kruskal

Thuật toán sẽ xây dựng tập cạnh *T* của cây khung nhỏ nhất *H*=<*V*, *T*> theo từng bước được mô tả trong Hình 5.5 như dưới đây.

a) Mô tả thuật toán

```
Thuật toán Kruskal:
Begin
    Bước 1 (Khởi tạo):
        T = ∅; //Khởi tạo tập cạnh cây khung là ∅
        d(H) = 0' //Khởi tạo độ dài nhỏ nhất cây khung là 0
    Bước 2 (Sắp xếp):
        <Sắp xếp các cạnh của đồ thị theo thứ tự giảm dần của trọng số>;
    Bước 3 (Lặp):
        while (|T<n-1| && E≠∅ ) do { // Lặp nếu E≠∅ và |T|<n-1
            e = <Cạnh có độ dài nhỏ nhất>;
            E = E \ {e}; //Loại cạnh e ra khỏi đồ thị
            if (T∪{e} không tạo nên chu trình ) then {
                T = ∪{e}; // Kết nạp e vào tập cạnh cây khung
                d(H) = d(H) + d(e); //Độ dài của tập cạnh cây khung
            endif;
        endwhile;
    Bước 4 (Trả lại kết quả):
        if (|T| < n-1) then <Đồ thị không liên thông>;
        else
            Return( T, d(H));
end.
```

Hình 5.5. Thuật toán Kruskal tìm cây khung nhỏ nhất.

b) Kiểm nghiệm thuật toán

Ví dụ ta cần kiểm nghiệm thuật toán Kruskal trong Hình 5.5 trên đồ thị được biểu diễn dưới dạng ma trận kề như dưới đây. Thực hiện tuần tự các bước của thuật toán ta sẽ được kết quả như sau:

∞	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	∞	∞	5	5	∞	∞	∞	∞	∞	∞
1	2	∞	4	∞	5	∞	∞	∞	∞	∞	∞	∞
3	∞	4	∞	5	5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	5	∞	6	∞	∞	∞	6	∞	∞	∞
∞	5	5	5	6	∞	6	6	6	6	∞	∞	∞
∞	5	∞	∞	∞	6	∞	6	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	6	6	∞	7	∞	∞	7	7
∞	∞	∞	∞	∞	6	∞	7	∞	7	7	∞	∞
∞	∞	∞	∞	6	6	∞	∞	7	∞	7	7	∞
∞	∞	∞	∞	∞	∞	∞	∞	7	7	∞	8	∞
∞	∞	∞	∞	∞	∞	∞	7	∞	7	8	∞	8
∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	8	∞

Bước 1: T = ϕ; D(T) = 0;

Bước 2. Sắp xếp các cạnh theo thứ tự tăng dần của trọng số

Đầu	Cuối	Tr.Số
1	2	2
1	3	1
1	4	3
2	3	2
2	6	5
2	7	5
3	4	4
3	6	5
4	5	5
4	6	5
5	6	6
5	10	6
6	7	6
6	8	6
6	9	6
6	10	6
7	8	6
8	9	7
8	12	7
8	13	7
9	10	7
9	11	7
10	11	7
10	12	7
11	12	8
12	13	8

Đầu	Cuối	Tr.Số
1	3	1
1	2	2
2	3	2
1	4	3
3	4	4
2	6	5
2	7	5
3	6	5
4	5	5
4	6	5
5	6	6
5	10	6
6	7	6
6	8	6
6	9	6
6	10	6
7	8	6
8	9	7
8	12	7
8	13	7
9	10	7
9	11	7
10	11	7
10	12	7
11	12	8
12	13	8

Bước 3 (lặp) :

STT	Cạnh được xét	T ∪e
1	E \(1,3)	T = T∪(1,3); D(T) = 1
2	E = E\(1,2)	T = T∪(1,2) ; D(T) = 1+2 =3
3	E = E\(2,3)	*Tạo nên chu trình*
4	E = E\(1,4)	T = T∪(1,4); D(T) = 3 +3 =6
5	E = E\(3,4)	*Tạo nên chu trình*
6	E = E\(2,6)	T = T∪(2,6); D(T) = 6+5=11
7	E = E\(2,7)	T = T∪(2,7); D(T) = 11+5 =16
8	E = E\(3,6)	*Tạo nên chu trình*
9	E = E\(4,5)	T = T∪ (4,5); D(T) = 16+5 =21
10	E = E\(4,6)	*Tạo nên chu trình*
11	E = E\(5,6)	*Tạo nên chu trình*
12	E = E\(5,10)	T = T∪(5,10); D(T) = 21+6 =27
13	E = E\(6,7)	*Tạo nên chu trình*
14	E = E\(6,8)	T = T∪(6,8); D(T) = 27+6 =33
15	E = E\(6,9)	T = T∪(6,9); D(T) = 33+6 =39
16	E = E\(6,10)	*Tạo nên chu trình*
17	E = E\(7,8)	*Tạo nên chu trình*
18	E = E\(8,9)	*Tạo nên chu trình*
19	E = E\(8,12)	T = T∪(8,12); D(T) = 39+7 =46
20	E = E\(8,13)	T = T∪(8,13); D(T) = 46+7 =53
21	E = E\(9,10)	*Tạo nên chu trình*
22	E = E\(9,11)	T = T∪(9,11); D(T) = 53+7 =60
Bước lặp kết thúc vì |T|> N-1 =12		

Bước 4 : Trả lại kết quả:

T = { (1,3), (1,2), (1,4), (2,6), 2,7), (4,5), (5,10), (6,8),(6,9), (8,12), (8,13), (9,11) }

D(T) = 1 + 2 + 3 + 5 + 5 + 5 + 6 + 6 + 6 + 7 + 7 +7 = 60

c) Cài đặt thuật toán

Chương trình tìm cây khung nhỏ nhất theo thuật toán Kruskal cho đồ thị biểu diễn dưới dạng danh sách trọng số được thể hiện dưới đây với các thủ tục:

- Thủ tục Init(): đọc dữ liệu biểu diễn bằng danh sách trọng số.
- Thủ tục Heap(): sắp xếp các cạnh theo thứ tự tăng dần của trọng số bằng thuật toán Heap Sort.
- Thủ tục Find(), Union() : tìm và kiểm tra khi kết nào cạnh vào cây khung có tạo nên chu trình hay không.
- Thủ tục Result() : đưa ra tập cạnh và độ dài nhỏ nhất của cây khung.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#define MAX 50
#define    TRUE 1
#define FALSE   0
int n, m, minl, connect;
int          dau[500],cuoi[500], w[500];
int           daut[50], cuoit[50], father[50];
void Init(void){
    int i; FILE *fp;
    fp=fopen("baotrum1.in","r");
    fscanf(fp, "%d%d", &n,&m);
    printf("\n So dinh do thi:%d", n);
    printf("\n So canh do thi:%d", m);
    printf("\n Danh sach ke do thi:");
    for(i=1; i<=m;i++){
            fscanf(fp, "%d%d%d", &dau[i], &cuoi[i], &w[i]);
            printf("\n Canh %d: %5d%5d%5d", i, dau[i], cuoi[i], w[i]);
    }
    fclose(fp);getch();
}
void Heap(int First, int Last){
    int j, k, t1, t2, t3;
    j=First;
    while(j<=(Last/2)){
            if( (2*j)<Last && w[2*j + 1]<w[2*j])
                    k = 2*j +1;
            else
                    k=2*j;
            if(w[k]<w[j]){
                    t1=dau[j]; t2=cuoi[j]; t3=w[j];
                    dau[j]=dau[k]; cuoi[j]=cuoi[k]; w[j]=w[k];
                    dau[k]=t1; cuoi[k]=t2; w[k]=t3;  j=k;
            }
            else j=Last;
    }
}


int Find(int i){
    int tro=i;
```

```
    while(father[tro]>0) tro=father[tro];
    return(tro);
}
void Union(int i, int j){
    int x = father[i]+father[j];
    if(father[i]>father[j]) {father[i]=j;father[j]=x; }
    else {
        father[j]=i;   father[i]=x;
    }
}
void Krusal(void){
    int i, last, u, v, r1, r2, ncanh, ndinh;
    for(i=1; i<=n; i++)         father[i]=-1;
    for(i= m/2;i>0; i++)
        Heap(i,m);
    last=m; ncanh=0; ndinh=0;minl=0;connect=TRUE;
    while(ndinh<n-1 && ncanh<m){
        ncanh=ncanh+1;     u=dau[1]; v=cuoi[1];
        r1= Find(u); r2= Find(v);
        if(r1!=r2) {
            ndinh=ndinh+1; Union(r1,r2);
            daut[ndinh]=u; cuoit[ndinh]=v;
            minl=minl+w[1];
        }
        dau[1]=dau[last];   cuoi[1]=cuoi[last]; w[1]=w[last]; last=last-1;
        Heap(1, last);
    }
    if(ndinh!=n-1) connect=FALSE;
}
void Result(void){
    int i;
    printf("\n Do dai cay khung nho nhat:%d", minl);
    printf("\n Cac canh cua cay khung nho nhat:");
    for(i=1; i<n; i++)
        printf("\n %5d%5d",daut[i], cuoit[i]);
 }
void main(void){
    Init(); Krusal();Result(); getch();
}
```

5.4.2. Thuật toán Prim

Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị có số cạnh khoảng m=n(n-1)/2. Trong những tình huống như vậy, thuật toán Prim tỏ ra hiệu quả hơn.

a) Mô tả thuật toán

Thuật toán Prim còn được mang tên là người láng giềng gần nhất. Trong thuật toán này, bắt đầu tại một đỉnh tuỳ ý *s* của đồ thị, nối *s* với đỉnh *y* sao cho trọng số cạnh *c[s, y]* là nhỏ nhất. Tiếp theo, từ đỉnh *s* hoặc *y* tìm cạnh có độ dài nhỏ nhất, điều này dẫn đến đỉnh thứ ba *z* và ta thu được cây bộ phận gồm *3* đỉnh *2* cạnh. Quá trình được tiếp tục cho tới khi ta nhận được cây gồm *n-1* cạnh, đó chính là cây bao trùm nhỏ nhất cần tìm. Thuật toán Prim được mô tả trong Hình 5.6.

Thuật toán PRIM (s):
Begin:
Bước 1 (Khởi tạo):
V_H = {*s*}; //Tập đỉnh cây khung thiết lập ban đầu là s
V = V\{s}; //Tập đỉnh V được bớt đi s
T = ∅; //Tập cạnh cây khung thiết lập ban đầu là ∅
d(H) = 0; //Độ dài cây khung được thiết lập là 0
Bước 2 (Lặp):
while (V≠ ∅) do {
e = <u, v>: cạnh có độ dài nhỏ nhất thỏa mãn u∈V, v∈V_H;
d(H) = d(H) + d(e); // Thiết lập đồ dài cây khung nhỏ nhất
T = T ∪ {e}; //Kết nạp e vào cây khung
V = V \{u}; // Tập đỉnh V bớt đi đỉnh u
V_H = V_H∪{u}; // Tập đỉnh V_H thêm vào đỉnh u
endwhile;
Bước 3 (Trả lại kết quả):
if (|T|<n-1) then <Đồ thị không liên thông>;
else Return(T, d(H));
End.

Hình 5.6. Thuật toán PRIM xây dựng cây khung nhỏ nhất.

b) Kiểm nghiệm thuật toán

Giả sử ta cần kiểm nghiệm thuật toán cho đồ thị trọng số Mục 5.4.1. Khi đó các bước thực hiện theo thuật toán PRIM như trong Bảng dưới đây.
Bước khởi tạo: T =ϕ; D(T)=0; V = 2,3,4,5,6,7,8,9,10,11,12,13; V_H =1

e=(v,t)| v∈V, t∈V_T có độ dài nhỏ nhất	V \v = ?	V_H ∪v=?	T, D(T)
(1,3)	2,4,5,6,7,8,9,10,11,12,13	1,3	T = T∪(1,3) D(T) = 0 +1
(1,2)	4,5,6,7,8,9,10,11,12,13	1,2,3	T = T∪(1,2) D(T) = 1+2=3
(1,4)	5,6,7,8,9,10,11,12,13	1,2,3,4	T = T∪(1,4) D(T) = 3+3=6
(2,6)	5, 7,8,9,10,11,12,13	1,2,3,4,6	T = T∪(2,6) D(T) = 6+5=11
(2,7)	5, 8,9,10,11,12,13	1,2,3,4,6,7	T = T∪(2,7) D(T) = 11+5=16
(4,5)	8,9,10,11,12,13	1,2,3,4,5, 6,7	T = T∪(4,5) D(T) = 16+5=21
(5,10)	8,9,11,12,13	1,2,3,4,5, 6,7,10	T = T∪(5,10) D(T) = 21+6=27
(6,8)	9,11,12,13	1,2,3,4,5, 6,7,8,10	T = T∪(6,8) D(T) = 27+6=33
(6,9)	11,12,13	1,2,3,4,5, 6,7,8,9,10	T = T∪(6,9) D(T) = 33+6=39
(8,12)	11,13	1,2,3,4,5, 6,7,8,9,10,12	T = T∪(8,12) D(T) = 39+7=46
(8,13)	11	1,2,3,4,5, 6,7,8,9,10,12,13	T = T∪(8,13) D(T) = 46+7=53
(9,11)	ϕ	1,2,3,4,5, 6,7,8,9,10,12,13,11	T = T∪(9,11) D(T) = 53+7=60
V = ϕ : kết thúc bước lặp			

Kết quả: T = { (1,3), (1,2), (1,4), (2,6), 2,7), (4,5), (5,10), (6,8),(6,9), (8,12), (8,13), (9,11) }
D(T) = 1 + 2 + 3 + 5 + 5 + 5 + 6 + 6 + 6 + 7 + 7 +7 = 60

c) Cài đặt thuật toán

Chương trình tìm cây khung nhỏ nhất theo thuật toán PRIM cho đồ thị biểu diễn dưới dạng danh sách trọng số được thể hiện dưới đây với các thủ tục:

- Thủ tục Init(): đọc dữ liệu biểu diễn bằng danh sách trọng số.
- Thủ tục Prim: Thuật toán PRIM xây dựng cây khung nhỏ nhất.
- Thủ tục Result() : đưa ra tập cạnh và độ dài nhỏ nhất của cây khung.

Chương trình cài đặt thuật toán Prim tìm cây bao trùm nhỏ nhất được thực hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#define         TRUE 1
#define FALSE        0
#define MAX 10000
int   a[100][100];
int  n,m, i,sc,w;
int   chuaxet[100];
int   cbt[100][3];
FILE *f;
void Init (void){
        int  p,i,j,k;
        for(i=1; i<=n; i++)
                for(j=1; j<=n;j++)
                        a[i][j]=0;
        f=fopen("baotrum.in","r");
        fscanf(f,"%d%d",&n,&m);
        printf("\n So dinh: %3d  ",n);
        printf("\n So canh: %3d", m);
        printf("\n Danh sach canh:");
        for(p=1; p<=m; p++){
                fscanf(f,"%d%d%d",&i,&j,&k);
                printf("\n %3d%3d%3d", i, j, k);
                a[i][j]=k; a[j][i]=k;
        }
        for (i=1; i<=n; i++){
                printf("\n");
                for (j=1; j<=n; j++){
                        if (i!=j && a[i][j]==0)
                                a[i][j]=MAX;
                        printf("%7d",a[i][j]);
                }
        }
        fclose(f);getch();
}
void Result(void){
         for(i=1;i<=sc; i++)
                printf("\n %3d%3d", cbt[i][1], cbt[i][2]);
}
void PRIM(void){
        int i,j,k,top,min,l,t,u;
        int s[100];
```

```
        sc=0;w=0;u=1;
        for(i=1; i<=n; i++)
                chuaxet[i]=TRUE;
        top=1;s[top]=u;
        chuaxet[u]=FALSE;
        while (sc<n-1) {
                min=MAX;
                for (i=1; i<=top; i++){
                        t=s[i];
                        for(j=1; j<=n; j++){
                                if (chuaxet[j] && min>a[t][j]){
                                        min=a[t][j];
                                        k=t;l=j;
                                }
                        }
                }
                sc++;w=w+min;
                cbt[sc][1]=k;cbt[sc][2]=l;
                chuaxet[l]=FALSE;a[k][l]=MAX;
                a[l][k]=MAX;top++;s[top]=l;
                printf("\n");
        }
}
void main(void){
        Init();PRIM();Result();
}
```

5.5. Những nội dung cần ghi nhớ

- ✓ Cây là đồ thị vô hướng liên thông không có chu trình. Do vậy, mọi đồ thị vô hướng liên thông đều có ít nhất một cây khung của nó.
- ✓ Hiểu cách biểu diễn và cài đặt được các loại cây: cây nhị phân tìm kiếm, cây quyết định, cây mã tiền tố và cây mã Huffman.
- ✓ Nắm vững phương pháp xây dựng cây khung của đồ thị bằng hai thuật toán duyệt theo chiều rộng và duyệt theo chiều sâu.
- ✓ Hiểu và cài đặt được các thuật toán Kruskal và Prim tìm cây bao trùm nhỏ nhất.

BÀI TẬP

1. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán xây dựng một cây khung của đồ thị bắt đầu tại đỉnh u∈V dựa vào thuật toán BFS(u)?

b) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

2. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán xây dựng một cây khung của đồ thị bắt đầu tại đỉnh u∈V dựa vào thuật toán DFS(u)?

b) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

3. Cho đồ thị vô hướng được biểu diễn dưới dạng danh sách kề như dưới đây

Ke(1) = { 2, 3, 4, 5 }.
Ke(2) = { 1, 3, 4 }.
Ke(3) = { 1, 2, 4 }.
Ke(4) = { 1, 2, 3 }.
Ke(5) = { 1, 6, 7, 8, 9 }.
Ke(6) = { 5, 7, 9 }.
Ke(7) = { 5, 6, 8, 10 }.
Ke(8) = { 5, 7, 9 }.
Ke(9) = { 5, 6, 8 }.
Ke(10) = { 7, 11, 12, 13 }.
Ke(11) = { 10, 12, 13 }.
Ke(12) = { 10, 11, 13 }.
Ke(13) = { 10, 11, 12 }.

Hãy thực hiện:

a) Trình bày thuật toán xây dựng cây khung của đồ thị bắt đầu tại đỉnh u dựa vào thuật toán DFS?

b) Xây dựng cây khung của đồ thị bắt đầu tại đỉnh u=3? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình xây dựng cây khung của đồ thị bắt đầu tại đỉnh u∈V?

4. Cho đồ thị vô hướng được biểu diễn dưới dạng danh sách kề như dưới đây

Ke(1) = { 2, 3, 4, 5 }.	Ke(5) = { 1, 6, 7, 8, 9 }.	Ke(9) = { 5, 6, 8 }.
Ke(2) = { 1, 3, 4 }.	Ke(6) = { 5, 7, 9 }.	Ke(10) = { 7, 11, 12, 13 }.
Ke(3) = { 1, 2, 4 }.	Ke(7) = { 5, 6, 8, 10 }.	Ke(11) = { 10, 12, 13 }.
Ke(4) = { 1, 2, 3 }.	Ke(8) = { 5, 7, 9 }.	Ke(12) = { 10, 11, 13 }.
		Ke(13) = { 10, 11, 12 }.

Hãy thực hiện:

a) Trình bày thuật toán xây dựng cây khung của đồ thị bắt đầu tại đỉnh u dựa vào thuật toán DFS?

b) Xây dựng cây khung của đồ thị bắt đầu tại đỉnh u=3? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình xây dựng cây khung của đồ thị bắt đầu tại đỉnh u∈V?

5. Cho đồ thị vô hướng có trọng số G =<V,E> được biểu diễn dưới dạng ma trận trọng số như hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán Prim tìm cây khung nhỏ nhất trên đồ thị vô hướng có trọng số?

b) Áp dụng thuật toán, tìm cây khung nhỏ nhất tại đỉnh số 1 của đồ thị G, chỉ rõ kết quả theo từng bước thực hiện của thuật toán?

c) Viết chương trình tìm cây khung nhỏ nhất của đồ thị bằng thuật toán PRIM?

∞	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	∞	∞	5	5	∞	∞	∞	∞	∞	∞
1	2	∞	4	∞	5	∞	∞	∞	∞	∞	∞	∞
3	∞	4	∞	5	5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	5	∞	6	∞	∞	∞	6	∞	∞	∞
∞	5	5	5	6	∞	6	6	6	6	∞	∞	∞
∞	5	∞	∞	∞	6	∞	6	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	6	6	∞	7	∞	∞	7	7
∞	∞	∞	∞	∞	6	∞	7	∞	7	7	∞	∞
∞	∞	∞	∞	6	6	∞	∞	7	∞	7	7	∞
∞	∞	∞	∞	∞	∞	∞	∞	7	7	∞	8	∞
∞	∞	∞	∞	∞	∞	∞	7	∞	7	8	∞	8
∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	8	∞

6. Cho đồ thị vô hướng có trọng số G =<V,E> được biểu diễn dưới dạng ma trận trọng số như hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán Kruskal tìm cây khung nhỏ nhất trên đồ thị vô hướng có trọng số?

b) Áp dụng thuật toán, tìm cây khung nhỏ nhất của đồ thị G, chỉ rõ kết quả theo từng bước thực hiện của thuật toán?

c) Viết chương trình tìm cây khung nhỏ nhất của đồ thị bằng thuật toán Kruskal?

∞	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	∞	∞	5	5	∞	∞	∞	∞	∞	∞
1	2	∞	4	∞	5	∞	∞	∞	∞	∞	∞	∞
3	∞	4	∞	5	5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	5	∞	6	∞	∞	∞	6	∞	∞	∞
∞	5	5	5	6	∞	6	6	6	6	∞	∞	∞
∞	5	∞	∞	∞	6	∞	6	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	6	6	∞	7	∞	∞	7	7
∞	∞	∞	∞	∞	6	∞	7	∞	7	7	∞	∞
∞	∞	∞	∞	6	6	∞	∞	7	∞	7	7	∞
∞	∞	∞	∞	∞	∞	∞	∞	7	7	∞	8	∞
∞	∞	∞	∞	∞	∞	∞	7	∞	7	8	∞	8
∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	8	∞

CHƯƠNG 6. BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT

Trong chương này chúng ta sẽ đề cập đến bài toán tìm đường đi ngắn nhất trên đồ thị. Đây là một trong những bài toán có ý nghĩa về lý thuyết và thực tế. Bạn đọc có thể tìm hiểu thêm về phương pháp chứng minh tính đúng đắn cũng như độ phức tạp của các thuật toán thông qua tài liệu [1, 2].

6.1. Phát biểu bài toán

Xét đồ thị $G=<V, E>$; trong đó $|V| = n$, $|E| = m$. Với mỗi cạnh $(u, v)\in E$, ta đặt tương ứng với nó một số thực A[u][v] được gọi là trọng số của cạnh. Ta sẽ đặt $A[u,v]=\infty$ nếu $(u, v)\notin E$. Nếu dãy $v_0, v_1, \ldots, v_k$ là một đường đi trên G thì $\sum_{i=1}^{p} A[v_{i-1}, v_i]$ được gọi là độ dài của đường đi.

Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể được phát biểu dưới dạng sau: tìm đường đi ngắn nhất từ một đỉnh xuất phát $s\in V$ (đỉnh nguồn) đến đỉnh cuối $t\in V$ (đỉnh đích). Đường đi như vậy được gọi là đường đi ngắn nhất từ *s* đến *t*, độ dài của đường đi *d(s,t)* được gọi là khoảng cách ngắn nhất từ *s* đến *t* (trong trường hợp tổng quát *d(s,t)* có thể âm). Nếu như không tồn tại đường đi từ *s* đến *t* thì độ dài đường đi $d(s,t)=\infty$. Dưới đây là một số thể hiện cụ thể của bài toán.

Trường hợp 1. Nếu *s* cố định và *t* thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất từ *s* đến tất cả các đỉnh còn lại trên đồ thị. Đối với đồ thị có trọng số không âm, bài toán luôn có lời giải bằng thuật toán Dijkstra. Đối với đồ thị có trọng số âm nhưng không tồn tại chu trình âm, bài toán có lời giải bằng thuật toán Bellman-Ford. Trong trường hợp đồ thị có chu trình âm, bài toán không có lời giải.

Trường hợp 2. Nếu s thay đổi và t cũng thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị. Bài toán luôn có lời giải trên đồ thị không có chu trình âm. Đối với đồ thị có trọng số không âm, bài toán được giải quyết bằng cách thực hiện lặp lại n lần thuật toán Dijkstra. Đối với đồ thị không có chu trình âm, bài toán có thể giải quyết bằng thuật toán Floyed.

Các thuật toán cụ thể giải quyết bài toán tìm đường đi ngắn nhất được thực hiện như dưới đây.

6.2. Thuật toán Dijkstra

Thuật toán tìm đường đi ngắn nhất từ đỉnh *s* đến các đỉnh còn lại được Dijkstra đề nghị áp dụng cho trường hợp đồ thị có hướng với trọng số không âm. Thuật toán được thực hiện trên cơ sở gán tạm thời cho các đỉnh. Nhãn của mỗi đỉnh cho biết cận trên của độ dài đường đi ngắn nhất tới đỉnh đó. Các nhãn này sẽ được biến đổi (tính lại) nhờ một

thủ tục lặp, mà ở mỗi bước lặp một số đỉnh sẽ có nhãn không thay đổi, nhãn đó chính là độ dài đường đi ngắn nhất từ s đến đỉnh đó.

6.2.1. Mô tả thuật toán

Thuật toán Dijkstra tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại của đồ thị được mô tả chi tiết trong Hình 6.1.

```
Thuật toán Dijkstra (s): //s∈ V là một đỉnh bất kỳ của G = <V,E>
Begin
    Bước 1 (Khởi tạo):
        d[s]=0; //Gán nhãn của đỉnh s là 0
        T = V\{s}; // T là tập đỉnh có nhãn tạm thời
        for each  v∈ V do { //Sử dụng s gán nhãn cho các đỉnh còn lại
            d[v] = A[s,v];
            truoc[v]=s;
        endfor;
    Bước 2 (Lặp):
        while (T≠ ∅ ) do {
            Tìm đỉnh u∈T sao cho d[u] = min { d[z] | z∈T};
            T= T\{u}; //cố định nhãn đỉnh u
            for each v∈ T do { //Sử dụng u, gán nhãn lại cho các đỉnh
                if ( d[v] > d[u] + A[u, v] ) then {
                    d[v] =  d[u] + A[u, v]; //Gán lại nhãn cho đỉnh v;
                    truoc[v] = u;
                endif;
            endfor;
        endwhlie;
    Bước 3 (Trả lại kết quả):
        Return (d[s], truoc[s]);
End.
```

Hình 6.1. Thuật toán Dijkstra.

6.2.2. Kiểm nghiệm thuật toán

Đầu vào của thuật toán :

- Ma trận trọng số không âm $A[u,v] = \begin{cases} d(u,v) & if \quad (u,v) \in E \\ \infty & if \quad (u,v) \notin E \end{cases}$

- s là đỉnh bất kỳ của đồ thị.

Ví dụ ta cần kiểm nghiệm thuật toán cho đồ thị được biểu diễn dưới dạng ma trận trọng số dưới đây. Khi đó, các bước thực hiện theo thuật toán Dijkstra tại đỉnh s =1 được thể hiện như Bảng 6.1.

∞	2	8	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	2	∞	∞	∞	9	∞	∞	∞	∞	∞	∞
∞	∞	∞	6	∞	8	1	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	1	7	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	1	∞	∞	9	8	∞	∞	∞	∞
∞	∞	∞	∞	∞	2	∞	2	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	9	∞	∞	2	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	6	∞	9	8
∞	∞	∞	∞	7	6	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	6	7	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	2
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	7	∞	∞

Bảng 6.1. Các bước thực hiện thuật toán Dijkstra tại s =1

Bước	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6	Đỉnh 7	Đỉnh 8	Đỉnh 9	Đỉnh 10	Đỉnh 11	Đỉnh 12	Đỉnh 13
1	<0,1>	<2,1>	<8,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
2	*	<2,1>	<4,2>	<∞,1>	<∞,1>	<∞,1>	<11,2>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
3	*	*	<4,2>	<10,3>	<∞,1>	<12,3>	<5,3>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
4	*	*	*	<10,3>	<∞,1>	<7, 7>	<5,3>	<7, 7>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
5	*	*	*	<10,3>	<8,6>	<7, 7>	*	<7,7>	<15,6>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
6	*	*	*	<10,3>	<8,6>	*	*	<7,7>	<15,6>	<∞,1>	<∞,1>	<9,8>	<∞,1>
7	*	*	*	<10,3>	<8,6>	*	*	*	<15,6>	<∞,1>	<∞,1>	<9,8>	<∞,1>
8	*	*	*	<10,3>	*	*	*	*	<15,6>	<∞,1>	<∞,1>	<9,8>	<11,12>
9	*	*	*	<10,3>	*	*	*	*	<15,6>	<∞,1>	<∞,1>	*	<11,12>
10	*	*	*	*	*	*	*	*	<15,6>	<∞,1>	<18,13>	*	<11,12>
11	*	*	*	*	*	*	*	*	<15,6>	<21,9>	<18,13>	*	*
12	*	*	*	*	*	*	*	*	*	<21,9>	<18,13>	*	*
13	*	*	*	*	*	*	*	*	*	<21,9>	*	*	*

Kết quả :

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 2: 2. Đường đi: 1-2.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3: 4. Đường đi: 1-2-3.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 4: 10. Đường đi: 1-2-3-10.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 5: 8. Đường đi: 1-2-3-7-6-5.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 6: 7. Đường đi: 1-2-3-7-6.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 7: 5. Đường đi: 1-2-3-7.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 8: 7. Đường đi: 1-2-3-7-8.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 9: 15. Đường đi: 1-2-3-7-6-9.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 10: 21. Đường đi: 1-2-3-7-6-9-10.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 11: 18. Đường đi: 1-2-3-7-8-12-13-11.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 12: 18. Đường đi: 1-2-3-7-8-12.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 13: 11. Đường đi: 1-2-3-7-8-12-13.

6.2.3. Cài đặt thuật toán

Chương trình cài đặt thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh khác của đồ thị có hướng với trọng số không âm được thực hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define     TRUE 1
#define FALSE    0
int n, s, t;
char chon;
int truoc[MAX], d[MAX], CP[MAX][MAX];
int   final[MAX];
void Init(void){
      FILE * fp;int i, j;
      fp = fopen("ijk1.in","r");
      fscanf(fp,"%d", &n);
      printf("\n So dinh :%d",n);
      printf("\n Ma tran khoang cach:");
      for(i=1; i<=n;i++){
            printf("\n");
            for(j=1; j<=n;j++){
                  fscanf(fp, "%d", &CP[i][j]);
                  printf("%3d",CP[i][j]);
                  if(CP[i][j]==0) CP[i][j]=32000;
            }
      }
      fclose(fp);
}
```

```
void Result(void){
        int i,j;
        printf("\n Duong di ngan nhat tu %d den %d la\n", s,t);
        printf("%d<=",t);
        i=truoc[t];
        while(i!=s){
                printf("%d<=",i);
                i=truoc[i];
        }
        printf("%d",s);
        printf("\n Do dai duong di la:%d", d[t]);
        getch();
}
void Dijkstra(void){
        int v, u, minp;
        printf("\n Tim duong di tu s=");scanf("%d", &s);
        printf("    den ");scanf("%d", &t);
        for(v=1; v<=n; v++){
                d[v]=CP[s][v];
                truoc[v]=s;
                final[v]=FALSE;
        }
        truoc[s]=0;   d[s]=0;final[s]=TRUE;
        while(!final[t]) {
                minp=2000;
                for(v=1; v<=n; v++){
                        if((!final[v]) && (minp>d[v]) ){
                                u=v;
                                minp=d[v];
                        }
                }
                final[u]=TRUE;// u- la dinh co nhan tam thoi nho nhat
                if(!final[t]){
                         for(v=1; v<=n; v++){
                                if ((!final[v]) && (d[u]+ CP[u][v]< d[v])){
                                        d[v]=d[u]+CP[u][v];
                                        truoc[v]=u;
                                }
                        }
                }

        }
}
```

```
void main(void){
        clrscr();Init();        Dijkstra();
        Result(); getch();
}
```

6.3.Thuật toán Bellman-Ford

Thuật toán Bellman-Ford dùng để tìm đường đi ngắn nhất trên đồ thị không có chu trình âm. Do vậy, trước khi thực hiện thuật toán Bellman-Ford ta cần kiểm tra đồ thị có chu trình âm hay không. Trong trường hợp đồ thị có chu trình âm, bài toán sẽ không có lời giải.

6.3.1. Mô tả thuật toán

Thuật toán được thực hiện theo k = n - 2 vòng lặp (n là số đỉnh của đồ thị) chi tiết trong Hình 6.2.

Thuật toán Bellman-Ford (s): //*s∈ V là đỉnh bất kỳ của đồ thị*
Begin:
Bước 1 (Khởi tạo):
for v∈V do { //*Sử dụng s gán nhãn cho các đỉnh v∈V*
D[v] = A[s][v];
Truoc[v] = s;
}
Bước 2 (Lặp) :
D[s] = 0; K=1;
while (K<=N-2) { //N-2 vòng lặp
for v∈V\{s} do { //*Lấy mỗi đỉnh v∈V\s*
for u∈V do { //*Gán nhãn cho v*
if (D[v] > D[u] + A[u][v]) {
D[v]= D[u] + A[u][v];
Truoc[v] = u;
endif;
endfor;
endfor;
endwlie;
Bước 3 (Trả lại kết quả):
Return(D[v], Truoc[v]: v∈U);
End.

Hình 6.2. Thuật toán Bellman-Ford.

6.3.2. Kiểm nghiệm thuật toán

Ví dụ ta cần kiểm nghiệm thuật toán Bellman-Ford cho đồ thị được biểu diễn dưới dạng ma trận trọng số sau:

$$A = \begin{vmatrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{vmatrix}$$

Khi đó, kết quả thực hiện theo thuật toán ta được kết quả sau:

Vòng lặp K=1:

v=2; D[2] = 1

D[1] + A[1, 2] = 0+1 (Không nhỏ hơn 1)

D[2] + A[2, 2] = 1 + ∞>1

D[3] + A[3, 2] = ∞ + ∞>1

D[4] + A[4, 2] = ∞ + ∞>1

D[5] + A[5, 2] = ∞ + ∞>1

v=3; D[3] = ∞

D[1] + A[1,3] = 0+∞

D[2] + A[2, 3] = 1 + 3 = 4<∞ (Thay D[3] = 4, Truoc[3] = 2)

D[3] + A[3, 3] = 4 + ∞>4

D[4] + A[4, 3] = ∞ + 2>4

D[5] + A[5, 3] = ∞ + ∞>4

v=4; D[4] = ∞

D[1] + A[1,4] = 0+∞

D[2] + A[2, 4] = 1 + 3 = 4<∞ (Thay D[4] = 4, Truoc[4] = 2)

D[3] + A[3, 4] = 4 + 1=5>4

D[4] + A[4, 4] = 4 + ∞ >4

D[5] + A[5, 4] = ∞ + 4>4

v=5; D[5] = 3

D[1] + A[1,5] = 0+3 (Không nhỏ hơn 3)

D[2] + A[2, 5] = 1 + 8 = 9>3

D[3] + A[3, 5] = 4 -5=-1<3 (Thay D[5] = -1, Truoc[5] =3)

D[4] + A[4, 5] = 4 + ∞ >-1

D[5] + A[5, 5] = -1 + ∞>-1

Vòng lặp K=2:

v=2; D[2] = 1

D[1] + A[1, 2] = 0+1 (Không nhỏ hơn 1)

D[2] + A[2, 2] = 1 + ∞>1

D[3] + A[3, 2] = 4 + ∞>1

D[4] + A[4, 2] = 4 + ∞>1

D[5] + A[5, 2] = -1 + ∞>1

v=3; D[3] = 4

D[1] + A[1, 3] = 0+∞>4

D[2] + A[2, 3] = 1 + 3 =4 (Không nhỏ hơn 4)

D[3] + A[3, 3] = 4 + ∞>4

D[4] + A[4, 3] = 4 + 2>4

D[5] + A[5, 3] = -1 + ∞>4

v=4; D[4] = 4

D[1] + A[1, 4] = 0+∞>4

D[2] + A[2, 4] = 1 + 3 =4 (Không nhỏ hơn 4)

D[3] + A[3, 4] = 4 + 1>4

D[4] + A[4, 4] = 4 + ∞>4

D[5] + A[5, 4] = -1 + 4=3< 4 (Thay D[4] = 5, Truoc[4] = 5

v=5; D[5] = -1

D[1] + A[1, 5] = 0+∞>-1

D[2] + A[2, 5] = 1 + 3 =-1

D[3] + A[3, 5] = 4 + 1>-1

D[4] + A[4, 5] = 3 + ∞>-1

D[5] + A[5, 5] = -1 + ∞>-1

Vòng lặp K=3:

v=2; D[2] = 1

D[1] + A[1, 2] = 0+1 (Không nhỏ hơn 1)

D[2] + A[2, 2] = 1 + ∞>1

D[3] + A[3, 2] = 4 + ∞>1

D[4] + A[4, 2] = 3 + ∞>1

D[5] + A[5, 2] = -1 + ∞>1

v=3; D[3] = 4

D[1] + A[1, 3] = 0+∞>4

D[2] + A[2, 3] = 1 + 3 =4 (Không nhỏ hơn 4)

D[3] + A[3, 3] = 4 + ∞>4

D[4] + A[4, 3] = 3 + 2>4

D[5] + A[5, 3] = -1 + ∞>4

v=4; D[4] = 3

D[1] + A[1, 4] = 0+∞>3

D[2] + A[2, 4] = 1 + 3 =3

D[3] + A[3, 4] = 4 + 1>3
D[4] + A[4, 4] = 3 + ∞>3
D[5] + A[5, 4] = -1 + 4=3(Không nhỏ hơn 3)

v=5; D[5] = -1
D[1] + A[1, 5] = 0+∞>-1
D[2] + A[2, 5] = 1 + 3 =-1
D[3] + A[3, 5] = 4 + 1>-1
D[4] + A[4, 5] = 3 + ∞>-1
D[5] + A[5, 5] = -1 + ∞>-1

Kết quả cuối cùng ta nhận được Bảng 6.2 dưới đây.

Bảng 6.2. Kết quả kiểm nghiệm theo thuật toán Bellman-Ford

K=?	D[1], Truoc[1]	D[2], Truoc[2]	D[3], Truoc[3]	D[4], Truoc[4]	D[5], Truoc[5]
	<0,1>	<1,1>	<∞,1>	<∞,1>	<3,1>
1	<0,1>	<1,1>	<4,2>	<4,2>	<-1,3>
2	<0,1>	<1,1>	<4,2>	<3,5>	<-1,3>
3	<0,1>	<1,1>	<4,2>	<3,5>	<-1,3>

6.3.3. Cài đặt thuật toán

Chương trình cài đặt thuật toán Bellman-Ford tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh khác của đồ thị có hướng, không có chu trình âm được thực hiện như sau:

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define MAX  100
#define  MAXC 10000

int C[MAX][MAX]; //Ma tran trong so bieu dien do thi
int D[MAX];      //Do dai duong di
int Trace[MAX];   //Luu lai vet duong di
int n, m, S, F;     // n:So dinh; S: Dinh bat dau; F:
Dinh ket thuc
FILE *fp;
void Read_Data(void){
     int i, u, v;fp = fopen("dothi.in","r");
     fscanf(fp,"%d%d%d%d",&n,&m,&S,&F);
     for(u=1; u<=n; u++)
        for(v=1; v<=n; v++)
```

```
            if (u==v) C[u][v]=0;
            else C[u][v]=MAXC;
      for(i=1; i<=m; i++)
          fscanf(fp,"%d%d%d",&u,&v,&C[u][v]);
      fclose(fp);
}
void Init(void){
    int i;
    for( i=1; i<=n; i++){
        D[i] = C[S][i];
        Trace[i]=S;
    }
}
void Result(void){
    if (D[F]==MAXC) printf("\n Khong co duong di");
    else {
         printf("\n Do dai %d den %d: %d", S, F, D[F]);
         while (F!=S ){
               printf("%d <--",F);
               F = Trace[F];
         }
    }
}
void Ford_Bellman(void){
    int k, u, v;D[S]=0;
    for( k=1; k<=n-2; k++){
         for(v=1; v<=n; v++){
            // if (v!=S ){
                for( u=1; u<=n; u++){
                     if (D[v]>D[u]+C[u][v]){
                        D[v] = D[u]+C[u][v];
                        Trace[u]=v;
                     }
                }
            // }
         }
    }
}
int main()
{
      Read_Data();Init();
      Ford_Bellman(); Result();
      system("PAUSE");
      return 0;
```

}

6.4.Thuật toán Floy

Để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị, chúng ta có thể sử dụng *n* lần thuật toán *Ford_Bellman* hoặc *Dijkstra* (trong trường hợp trọng số không âm). Tuy nhiên, trong cả hai thuật toán được sử dụng đều có độ phức tạp tính toán lớn (chí ít là $O(n^3)$). Trong trường hợp tổng quát, người ta thường dùng thuật toán *Floy*.

6.4.1. Mô tả thuật toán

Thuật toán Floy được mô tả chi tiết trong Hình 6.3.

```
Thuật toán Floy:
Begin:
    Bước 1 (Khởi tạo):
        for (i=1; i≤ n; i++) {
            for (j =1; j≤ n; j++) {
                d[i,j] = a[i, j];
                p[i,j] = i;
            }
        }
    Bước 2 (lặp) :
        for (k=1; k≤ n; k++) {
            for (i=1; i≤ n; i++){
                for (j =1; j≤ n; j++) {
                    if (d[i,j] > d[i, k] + d[k, j]) {
                        d[i, j] = d[i, k] + d[k, j];
                        p[i,j] = p[k, j];
                    }
                }
            }
        }
    }
    Bước 3 (Trả lại kết quả):
        Return (p([i,j], d[i,j]: i, j∈V);
```

Hình 6.3. Thuật toán Floy.

6.4.2. Cài đặt thuật toán

Chương trình cài đặt thuật toán Foly tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 10000
#define	TRUE 1
#define FALSE	0
int A[50][50], D[50][50], S[50][50];
int n, u, v, k;FILE *fp;
void Init(void){
	int i, j, k;
	fp=fopen("FLOY.IN","r");
	if(fp==NULL){
		printf("\n Khong co file input");
		getch(); return;
	}
	for(i=1; i<=n; i++)
		for(j=1; j<=n; j++)
			A[i][j]=0;
	fscanf(fp,"%d%d%d",&n,&u,&v);
	printf("\n So dinh do thi:%d",n);
	printf("\n Di tu dinh:%d den dinh %d:",u,v);
	printf("\n Ma tran trong so:");
	for(i=1; i<=n; i++){
		printf("\n");
		for(j=1; j<=n; j++){
			fscanf(fp,"%d", &A[i][j]);
			printf("%5d",A[i][j]);
			if(i!=j && A[i][j]==0)
				A[i][j]=MAX;
		}
	}
	fclose(fp);getch();
}
void Result(void){
	if(D[u][v]>=MAX) {
		printf("\n Khong co duong di");
		getch(); return;
```

```
        }
        else {
                printf("\n Duong di ngan nhat:%d", D[u][v]);
                printf("\n Dinh %3d", u);
                while(u!=v) {
                        printf("%3d",S[u][v]);
                        u=S[u][v];
                }
        }
}
void Floy(void){
        int i, j,k, found;
        for(i=1; i<=n; i++){
                for(j=1; j<=n;j++){
                        D[i][j]=A[i][j];
                        if (D[i][j]==MAX) S[i][j]=0;
                        else S[i][j]=j;
                }
        }
        /* Mang D[i,j] la mang chua cac gia tri khoan cach ngan nhat tu i den j
        Mang S la mang chua gia tri phan tu ngay sau cua i tren duong di
        ngan nhat tu i->j  */
        for (k=1; k<=n; k++){
                for (i=1; i<=n; i++){
                        for (j=1; j<=n; j++){
                                if (D[i][k]!=MAX && D[i][j]>(D[i][k]+D[k][j]) ){
                                // Tim D[i,j] nho nhat co the co
                                        D[i][j]=D[i][k]+D[k][j];
                                        S[i][j]=S[i][k];
                                        //ung voi no la gia tri cua phan tu ngay sau i
                                }
                        }
                }
        }
}
void main(void){
        clrscr();Init();
        Floy();Result();
}
```

6.5. Những nội dung cần ghi nhớ

- ✓ Hiểu bài toán tìm đường đi ngắn nhất và các dạng cụ thể của bài toán.
- ✓ Hiểu thuật toán, kiểm nghiệm thuật toán và cài đặt được thuật toán Dijkstra.
- ✓ Hiểu thuật toán, kiểm nghiệm thuật toán và cài đặt được thuật toán Bellman-Ford.
- ✓ Hiểu thuật toán, kiểm nghiệm thuật toán và cài đặt được thuật toán Floy.

BÀI TẬP

1. Cho đồ thị gồm 7 đỉnh cho bởi ma trận trọng số

00	11	65	17	65	65	65
65	00	12	65	65	10	16
65	65	00	13	14	65	19
65	65	65	00	65	65	18
65	65	65	65	00	65	15
65	13	18	65	65	00	10
65	65	65	65	65	65	00

Tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh 7. Yêu cầu chỉ rõ những kết quả trung gian trong quá trình thực hiện thuật toán.

2. Cho Cơ sở dữ liệu ghi lại thông tin về N ***Tuyến bay*** (N<=100) của một hãng hàng không. Trong đó, thông tin về mỗi tuyến bay được mô tả bởi: Điểm khởi hành (departure), điểm đến (destination), khoảng cách (lenght). Departure, destination là một xâu kí tự độ dài không quá 32, không chứa dấu trống ở giữa, Length là một số nhỏ hơn 32767.

Ta gọi "***Hành trình bay***" từ điểm khởi hành A tới điểm đến B là dãy các hành trình [A, A_1, n_1], [A_1, A_2, n_2] . . .[A_k, B,n_k] với A_i là điểm đến của tuyến i nhưng lại là điểm khởi hành của tuyến i +1, n_i là khoảng cách của tuyến bay thứ i (1<=i<k). Trong đó, khoảng cách của hành trình là tổng khoảng cách của các tuyến mà hành trình đi qua (n_1+n_2+. .+n_k).

Cho file dữ liệu kiểu text hanhtrinh.in được ghi theo từng dòng, số các dòng trong file dữ liệu không vượt quá N, trên mỗi dòng ghi lại thông tin về một tuyến bay, trong đó departure, destination, length được phân biệt với nhau bởi một hoặc vài dấu trống. Hãy tìm giải pháp để thoả mãn nhu cầu của khách hàng đi từ A đến B theo một số tình huống sau:

Tìm hành trình có khoảng cách bé nhất từ A đến B. In ra màn hình từng điểm mà hành trình đã qua và khoảng cách của hành trình. Nếu hành trình không tồn tại hãy đưa ra thông báo "Hành trình không tồn tại".

Ví dụ về Cơ sở dữ liệu hanhtrinh.in

New_York	Chicago	1000	
Chicago	Denver		1000
New_York	Toronto	800	
New_York	Denver		1900

Toronto	Calgary	1500
Toronto	Los_Angeles	1800
Toronto	Chicago	500
Denver	Urbana	1000
Denver	Houston	1500
Houston	Los_Angeles	1500
Denver	Los_Angeles	1000

Với điểm đi : New_York, điểm đến : Los_Angeles ; chúng ta sẽ có kết quả sau:

Hành trình ngắn nhất:

New_York to Toronto to Los_Angeles; Khoảng cách: 2600.

3. Kế tục thành công với khối lập phương thần bí, Rubik sáng tạo ra dạng phẳng của trò chơi này gọi là trò chơi các ô vuông thần bí. Đó là một bảng gồm 8 ô vuông bằng nhau như hình 1. Chúng ta qui định trên mỗi ô vuông có một màu khác nhau. Các màu được kí hiệu bởi 8 số nguyên tương ứng với tám màu cơ bản của màn hình EGA, VGA như hình 1. Trạng thái của bảng các màu được cho bởi dãy kí hiệu màu các ô được viết lần lượt theo chiều kim đồng hồ bắt đầu từ ô góc trên bên trái và kết thúc ở ô góc dưới bên trái. Ví dụ: trạng thái trong hình 1 được cho bởi dãy các màu tương ứng với dãy số (1, 2, 3, 4, 5 , 6, 7, 8). Trạng thái này được gọi là trạng thái khởi đầu.

Biết rằng chỉ cần sử dụng 3 phép biến đổi cơ bản có tên là 'A', 'B', 'C' dưới đây bao giờ cũng chuyển được từ trạng thái khởi đầu về trạng thái bất kỳ:

'A' : đổi chỗ dòng trên xuống dòng dưới. Ví dụ sau phép biến đổi A, hình 1 sẽ trở thành hình 2:

'B' : thực hiện một phép hoán vị vòng quanh từ trái sang phải trên từng dòng. Ví dụ sau phép biển đổi B hình 1 sẽ trở thành hình 3:

'C' : quay theo chiều kim đồng hồ bốn ô ở giữa. Ví dụ sau phép biến đổi C hình 1 trở thành hình 4:

Hình 1

1	2	3	4
8	7	6	5

Hình 2

8	7	6	5
1	2	3	4

Hình 3

4	1	2	3
5	8	7	6

Hình 4

1	7	2	4
8	6	3	5

Cho file dữ liệu Input.txt ghi lại 8 số nguyên trên một dòng, mỗi số được phân biệt với nhau bởi một dấu trống ghi lại trạng thái đích. Hãy tìm dãy các phép biến đổi sơ bản để đưa trạng thái khởi đầu về trạng thái đích sao cho số các phép biến đổi là ít nhất có thể được.

Dữ liệu ra được ghi lại trong file Output.txt, dòng đầu tiên ghi lại số các phép biến đổi, những dòng tiếp theo ghi lại tên của các thao tác cơ bản đã thực hiện, mỗi thao tác cơ bản được viết trên một dòng.

Bạn sẽ được thêm 20 điểm nếu sử dụng bảng màu thích hợp của màn hình để mô tả lại các phép biến đổi trạng thái của trò chơi. Ví dụ với trạng thái đích dưới đây sẽ cho ta kết quả như sau:

Input.txt	Output.txt
2 6 8 4 5 7 3 1	7 B C A B C C B

4. Cho một mạng thông tin gồm N nút. Trong đó, đường truyền tin hai chiều trực tiếp từ nút i đến nút j có chi phí truyền thông tương ứng là một số nguyên A[i,j] = A[j,i], với A[i,j]>=0, i ≠ j. Nếu đường truyền tin từ nút i_1 đến nút i_k phải thông qua các nút i_2, . . i_{k-1} thì chi phí truyền thông được tính bằng tổng các chi phí truyền thông A[i_1,i_2], A[i_2,i_3], . . . A[i_{k-1},i_k]. Cho trước hai nút i và j. Hãy tìm một đường truyền tin từ nút i đến nút j sao cho chi phí truyền thông là thấp nhất.

Dữ liệu vào được cho bởi file TEXT có tên INP.NN. Trong đó, dòng thứ nhất ghi ba số N, i, j, dòng thứ k + 1 ghi k-1 số A[k,1], A[k,2], . . , A[k,k-1], 1<=k<=N.
Kết quả thông báo ra file TEXT có tên OUT.NN. Trong đó, dòng thứ nhất ghi chi phí truyền thông thấp nhất từ nút i đến nút j, dòng thứ 2 ghi lần lượt các nút trên đường truyền tin có chi phí truyền thông thấp nhất từ nút i tới nút j.

5. Cho một mạng thông tin gồm N nút. Trong đó, đường truyền tin hai chiều trực tiếp từ nút i đến nút j có chi phí truyền thông tương ứng là một số nguyên A[i,j] = A[j,i], với A[i,j]>=0, i ≠ j. Nếu đường truyền tin từ nút i_1 đến nút i_k phải thông qua các nút i_2, . . i_{k-1} thì chi phí truyền thông được tính bằng tổng các chi phí truyền thông A[i_1,i_2], A[i_2,i_3], . . . A[i_{k-1},i_k]. Biết rằng, giữa hai nút bất kỳ của mạng thông tin đều tồn tại ít nhất một đường truyền tin.

Để tiết kiệm đường truyền, người ta tìm cách loại bỏ đi một số đường truyền tin mà vẫn đảm bảo được tính liên thông của mạng. Hãy tìm một phương án loại bỏ đi những đường truyền tin, sao cho ta nhận được một mạng liên thông có chi phí tối thiểu nhất có thể được.

Dữ liệu vào được cho bởi file TEXT có tên INP.NN. Trong đó, dòng thứ nhất ghi số N, dòng thứ k + 1 ghi k-1 số A[k,1], A[k,2], . . , A[k,k-1], 1<=k<=N.

Kết quả thông báo ra file TEXT có tên OUT.NN trong đó dòng thứ nhất ghi chi phí truyền thông nhỏ nhất trong toàn mạng. Từ dòng thứ 2 ghi lần lượt các nút trên đường truyền tin, mỗi đường truyền ghi trên một dòng.

5. Cho đồ thị có hướng có trọng số được biểu diễn dưới dạng ma trận trọng số như dưới đây. Hãy thực hiện:

a) Trình bày thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh $s \in V$ đến các đỉnh còn lại của đồ thị?

b) Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Tìm đường đi ngắn nhất từ đỉnh 5 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

d) Viết chương trình tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị?

∞	2	8	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	2	∞	∞	∞	9	∞	∞	∞	∞	∞	∞
∞	∞	∞	6	∞	8	1	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	1	7	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	1	∞	∞	9	8	∞	∞	∞	∞
∞	∞	∞	∞	∞	2	∞	2	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	9	∞	∞	2	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	6	∞	9	8
∞	∞	∞	∞	7	6	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	6	7	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	2
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	7	∞	∞

6. Cho đồ thị có hướng có trọng số được biểu diễn dưới dạng ma trận trọng số như dưới đây. Hãy thực hiện:

a) Trình bày thuật toán Bellman-Ford tìm đường đi ngắn nhất từ đỉnh $s \in V$ đến các đỉnh còn lại của đồ thị?

b) Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Tìm đường đi ngắn nhất từ đỉnh 5 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

d) Viết chương trình tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị?

∞	7	∞	9	4	∞	∞	∞	∞
∞	∞	3	∞	-4	∞	∞	∞	∞
∞	∞	∞	∞	-8	∞	-3	∞	∞
∞	∞	∞	∞	∞	∞	∞	-4	∞
∞	∞	∞	5	∞	2	∞	3	∞
∞	∞	∞	∞	∞	∞	5	∞	2
∞	∞	∞	∞	∞	∞	∞	∞	-7
∞	∞	∞	∞	∞	-2	∞	∞	-3
∞	∞	∞	∞	∞	∞	∞	∞	∞